

OPERATSION TIZIMLAR • 6-MA'RUZA

OTda ko'p vazifalilikni ta'minlash usullari

Jarayonlar, oqimlar, tolalar.

Jarayonlararo aloqa. Sinxronizatsiya

1. Ko'p vazifalilik: asosiy tushunchalar
2. Jarayon: model, holatlar, PCB
3. Hayot sikli: fork, exec, wait, exit
4. Windowsda jarayonlar
5. Protsessorni rejalashtirish
6. Linux va Windows rejalashtiruvchilari
7. Oqimlar (threads)
8. Tolalar (fibers) va korutinalar
9. Jarayonlararo aloqa (IPC)
10. Kanallar va signallar
11. Umumiy xotira, navbatlar, socketlar
12. Poyga holati va kritik seksiya
13. Sinxronizatsiya primitivlari
14. Klassik masalalar
15. O'zaro bloklanish (deadlock)
16. Kuzatish vositalari, amaliyot

1. Ko'p vazifalilik: asosiy tushunchalar

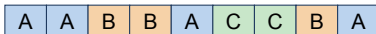
Ko'p vazifalilik nima?

Ta'rif

Ko'p vazifalilik (multitasking) — OTning bir nechta vazifani shunday bajarish xususiyatiki, protsessor bitta bo'lsa ham, har biri oldinga siljiydi.

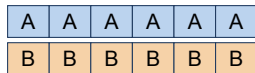
- Yadrolar: odatda 4–16 ta. Vazifalar: yuzlab.
- OT protsessorni tez almashtirib, **bir vaqtdalik illyuziyasini** yaratadi.

1 yadro:



Konkurentlik — navbatlashish

2 yadro:



Parallellik — haqiqatan bir vaqtda

Tarixiy rivojlanish. Kooperativ va siqib chiqaruvchi

Bosqich	G'oya	Hal qiladigan muammo
Paketli ishlov berish	Vazifalar qat'iy navbat bilan	Dasturlarni qo'lda yuklash
Multidasturlash	Biri I/O kutganda boshqasi ishlaydi	I/O da CPU ning bekor turishi
Vaqtning taqsimlash	Har biriga qisqa vaqt kvanti	Interaktivlik
Ko'p yadroliklik	Haqiqiy parallelizm	Chastotasiz unumdorlik o'sishi

Kooperativ

Vazifa CPU ni **o'zi** bo'shatadi (yield, I/O). Bitta cheksiz sikl — butun tizim qotadi. Windows 3.x, Mac OS 9 gacha.

Siqib chiqaruvchi

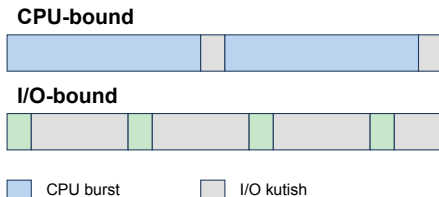
OT kvant tugaganda CPU ni **taymer uzilishi** orqali tortib oladi. Linux, Windows NT+, macOS.

CPU-bound va I/O-bound vazifalar

- **CPU-bound** — ko'p hisoblaydi (video siqish, kompilyatsiya): uzun CPU portlashlari
- **I/O-bound** — ko'p kutadi (muharrir, veb-server): qisqa CPU portlashlari

Qoida

Yaxshi rejalashtiruvchi I/O-bound vazifalarga ustuvorlik beradi: ular CPU ni tez bo'shatadi, foydalanuvchi tez javob oladi.



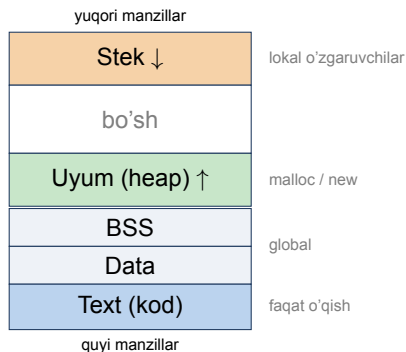
2. Jarayon: model, holatlar, PCB

Jarayon va uning manzil maydoni

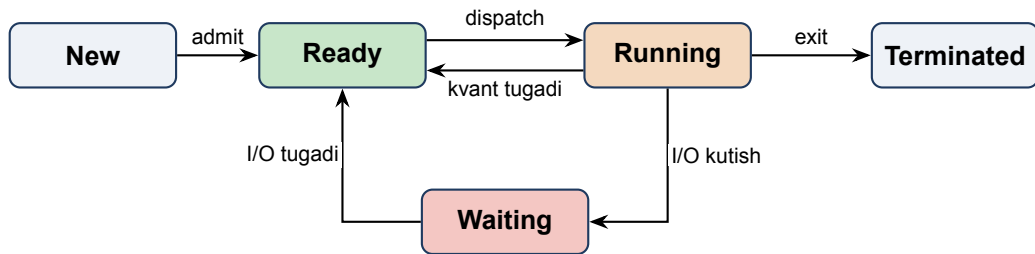
Ta'rif

Jarayon — bajarilish holatidagi dastur va OT ajratgan resurslar: manzil maydoni, ochiq fayllar, hisob ma'lumotlari, registrlar.

- **Dastur** — diskdagi passiv fayl (/usr/bin/python3)
- **Jarayon** — faol mohiyat; 3 terminal = 3 ta bash jarayoni
- Har bir jarayonda **o'z virtual manzil maydoni** — izolyatsiya



Jarayon holatlari modeli



- **Running** → **Ready** — siqib chiqarildi, ishlashi *mumkin edi*
- **Running** → **Waiting** — o'zi davom eta olmaydi, ma'lumot kutadi
- **Waiting** → **Running** to'g'ridan-to'g'ri **mumkin emas** — avval Ready ga
- Bitta yadroda Running holatida ko'pi bilan **bitta** jarayon

ps dagi STAT

R Running yoki Ready
S Uzilishi mumkin bo'lgan uyqu
D Uzilmaydigan uyqu (I/O)
T To'xtatilgan (Ctrl+Z)
Z Zombi

```
cat /proc/$$/status | head  
ls -l /proc/$$/fd
```

PCB (Process Control Block)

Linux: task_struct; Windows: EPROCESS

Identifikator	PID, PPID
Holat	Ready, Running...
CPU konteksti	PC, registrlar, stek ko'rsatkichi
Rejalashtirish	Ustuvorlik, nice, CPU vaqti
Xotira	Sahifalar jadvallari
Fayllar	Ochiq deskriptorlar jadvali
Xavfsizlik	UID, GID, capabilities

3. Jarayonning hayot sikli: fork, exec, wait, exit

fork() — bitta chaqiruv, ikkita qaytish

```
int x = 5;
pid_t pid = fork();
if (pid == -1) { perror("fork"); return 1; }
if (pid == 0) {                               // bola
    x += 10;
    printf("child: x=%d\n", x);
    exit(0);
}
wait(NULL);                                   // ota kutadi
printf("parent: x=%d\n", x);
```

```
child: x=15
parent: x=5
```

Qayerda

Qaytaradi

Ota-jarayonda	bola PID i (> 0)
Bola-jarayonda	0
Xatoda	-1

Nega otada $x = 5$?

fork() dan keyin bolada xotiraning **o'z nusxasi**. O'zgarishlar boshqasida ko'rinmaydi.

Copy-on-Write

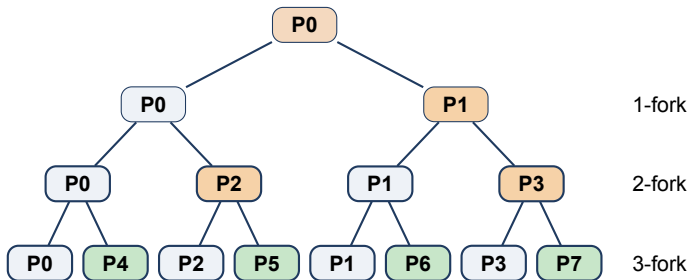
Sahifalar faqat **yozishda** nusxalanadi. Bola darhol exec() qilsa, deyarli hech narsa nusxalanmaydi.

Jarayonlarni sanash: n ta fork $\Rightarrow 2^n$

```
fork();  
fork();  
fork();  
printf("salom\n");
```

Formula

n ta ketma-ket fork() $\Rightarrow 2^n$
jarayon
(yangilari: $2^n - 1$)



8 jarayon $\Rightarrow$ 8 ta "salom"

&& va || o'ng qismini har doim hisoblamaydi

```
if (fork() && fork())  
    fork();  
printf("x\n");    // 4 jarayon
```

Ikkinchi fork() ni faqat **ota** bajaradi: bolada birinchi fork() 0 qaytaradi va && to'xtaydi.

Buferlash tuzog'i

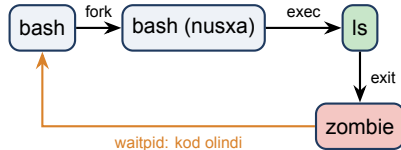
printf buferga yozadi. Chiqish kanal yoki faylga yo'naltirilsa, bufer fork() da **nusxalanadi** — satrlar takrorlanadi.

```
fflush(stdout);    // fork dan  
oldin  
setvbuf(stdout, NULL, _IONBF, 0); // yoki
```

exec() va qobiq buyruqni qanday ishga tushiradi

```
pid_t pid = fork();
if (pid == 0) {
    execlp("ls", "ls", "-l", "/tmp", NULL)
    ;
    perror("execlp"); // faqat xatoda
    _exit(127);
}
int status;
waitpid(pid, &status, 0);
if (WIFEXITED(status))
    printf("ls exited %d\n",
        WEXITSTATUS(status));
```

- exec kod, ma'lumot va stekni **almashtiradi**
- **PID saqlanadi**, ochiq fayllar ham
- Muvaffaqiyatda exec **qaytmaydi**



Vaziyat	Nomi	Nima bo'ladi
Bola tugadi, ota wait qilmagan	Zombi (Z)	Jadvalda faqat yozuv: PID va qaytish kodi. CPU va xotira sarflamaydi, lekin PID ni egallaydi
Ota boladan oldin tugadi	Yetim	Bolani systemd (PID 1) asrab oladi va o'zi wait qiladi

Zombini o'ldirib bo'lmaydi

U allaqachon o'lgan! `kill -9` yordam bermaydi. Ota wait qilishi yoki otani yakunlash kerak.

Nega xavfli?

Server minglab bola yaratib wait qilmasa, jarayonlar jadvali to'ladi — yangi jarayon yaratib bo'lmaydi.

4. Windowsda jarayonlar

CreateProcess va POSIX bilan taqqoslash

```
STARTUPINFO si = { sizeof(si) };  
PROCESS_INFORMATION pi;  
CreateProcess(L"C:\\...\\notepad.exe",  
             NULL, NULL, NULL, FALSE, 0,  
             NULL, NULL, &si, &pi);  
WaitForSingleObject(pi.hProcess,  
                    INFINITE);  
CloseHandle(pi.hThread);  
CloseHandle(pi.hProcess);
```

Get-Process, tasklist, taskkill /PID
4312 /F

	Linux	Windows
Yaratish	fork + exec	CreateProcess (1 qadam)
Kutish	waitpid	WaitForSingleObject
Identifikatsiya	PID	PID + HANDLE
Ierarxiya	Qat'iy daraxt	Zaif
Zombilar	wait gacha	HANDLE ochiq ekan
Guruhlash	cgroups	Job objects
Rejalashtirish birligi	Oqim (task)	Oqim

5. Protsessorni rejalashtirish

Rejalashtiruvchi Ready navbatidan jarayon tanlaydi; **dispatcher** almashtiradi.

Qaror qachon:

1. Running → Waiting
2. Running → Ready (taymer)
3. Waiting → Ready (I/O tugadi)
4. Jarayon tugadi

Faqat 1, 4 — **siqib chiqarmaydigan**; 2, 3 ham — **siqib chiqaruvchi**.

Mezon	Ta'rif	
CPU bandligi	CPU band vaqt ulushi	↑
Throughput	Vaqt birligida tugaganlar	↑
Aylanish vaqti	Tugash — Kelish	↓
Kutish vaqti	Aylanish — Burst	↓
Javob vaqti	1-ishga tushish — Kelish	↓

Rejalashtirish algoritmlari

Algoritm	Tamoyil	Siqadi?	+	—
FCFS	Kim oldin kelsa	Yo'q	Oddiy	Karvon effekti
SJF	Eng qisqa burst	Yo'q	Min. o'rtacha kutish (siqmaydiganlar orasida)	Davomiylik noma'lum; ochlik
SRTF	Siqadigan SJF	Ha	Umuman min. o'rtacha kutish	Ko'p almashtirish, ochlik
Ustuvorlik	Eng yuqori ustuvorlik	Ikkalasi	Muhimlar tezroq	Ochlik → qarish (aging)
RR	Aylana, kvant q	Ha	Yaxshi javob, ochliksiz	Katta o'rtacha aylanish
MLFQ	Ko'p navbat, pastga tushirish	Ha	I/O-bound ni o'zi ajratadi	Sozlash murakkab

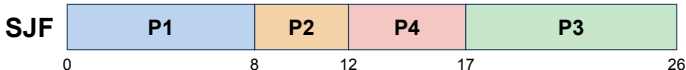
Misol: FCFS va SJF

	Kelish	Burst
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Kutish = Boshlanish –
Kelish
(siqmaydiganlarda)



Kutish: $0 + 7 + 10 + 18 = 35$; **o'rtacha = 8,75**

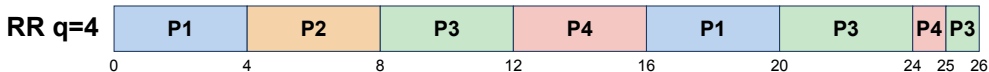


0 da faqat P1 bor. 8 da: $P2 (4) < P4 (5) < P3 (9)$
Kutish: $0 + 7 + 9 + 15 = 31$; **o'rtacha = 7,75**

Misol: SRTF va RR (q = 4)



1 da P2 (4) keladi: P1 da 7 qolgan $> 4 \Rightarrow$ P1 siqiladi. Kutish = Aylanish – Burst: P1 = $17 - 8 = 9$, P2 = $4 - 4 = 0$, P4 = $7 - 5 = 2$, P3 = $24 - 9 = 15$. **O'rtacha = 6,5**



Kutish: P1 = 12, P2 = 3, P3 = 15, P4 = 17. **O'rtacha = 11,75**

	FCFS	SJF	SRTF	RR
O'rtacha kutish	8,75	7,75	6,5	11,75
O'rtacha aylanish	15,25	14,25	13,0	18,25

Xulosa

SRTF — minimal o'rtacha kutish, lekin kelajakni bilish kerak.

RR — o'rtacha kutishda yutqazadi, lekin hech kim keyingi kvantigacha $(n - 1) \cdot q$ dan ko'p kutmaydi. Interaktiv tizimlarga aynan shu kerak.

Kvant q :

- **juda katta** $\rightarrow$ RR = FCFS, javob yomon
- **juda kichik** $\rightarrow$ vaqt almashtirishga ketadi
- **amalda**: almashtirishdan (mikrosekundlar) ancha katta; odatda 1–100 ms

6. Linux va Windows rejalashtiruvchilari

- **CFS** (2.6.23, 2007): eng kichik vruntime li vazifa; qizil-qora daraxt
- **EEVDF** (6.6, 2023): adolat + kechikish talablari
- **nice**: -20 (yuqori) ... +19 (past); kamaytirish — faqat root
- Real vaqt: SCHED_FIFO, SCHED_RR, ustuvorlik 1–99

```
nice -n 10 ./long_job
renice -n 5 -p 4312
chrt -p 4312
taskset -c 0 ./job      # CPU affinity
```

Windows

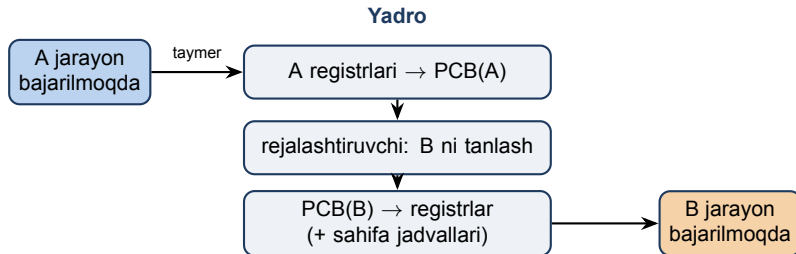
Oqimlar **32 daraja** bo'yicha:

0 — nollash oqimi

1–15 — dinamik (oddiy)

16–31 — real vaqt

Har daraja ichida siqadigan RR. Faol oyna va kiritishni kutgan oqim — **priority boost**.



Qo'shimcha xarajat

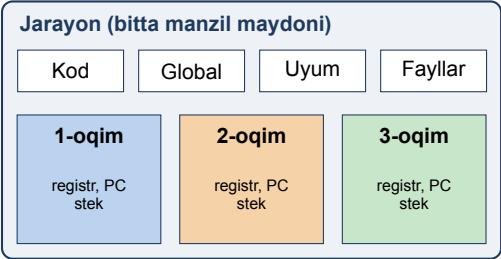
Bu vaqtda foydali ish yo'q; keshlar va TLB "soviydi". Oqimlar orasida almashtirish **arzonroq**: sahifa jadvallari o'zgarmaydi.

```
grep ctxt /proc/stat
grep ctxt_switches /proc/$$/status
vmstat 1 5 # cs ustuni
```

voluntary — o'zi bloklandi; **nonvoluntary** — siqib chiqarildi

7. Oqimlar (threads)

Oqim: nima umumiy, nima o'ziniki



Umumiy	O'ziniki
Kod	PC
Global, uyum	Registrlar
Ochiq fayllar	Stek
PID, UID	TID
Signal ishlovchi-lari	Signal maskasi, errno
	TLS

Nima uchun oqimlar? Jarayonmi yoki oqim?

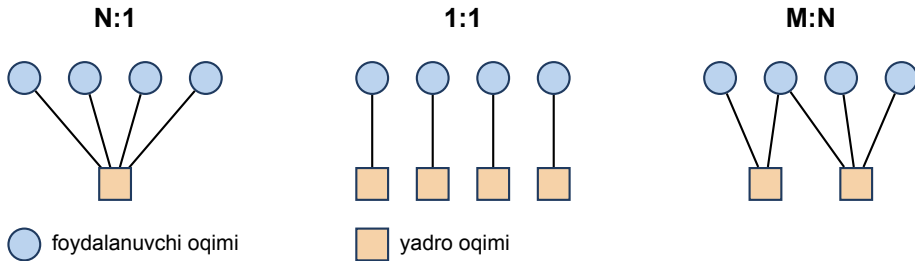
1. **Javob tezligi** — GUI javob beradi, ishchi yuklaydi
2. **Bo'lishish** — IPC kerak emas
3. **Tejamkorlik** — yaratish, almashtirish arzon
4. **Masshtab** — yadrolarda parallel

Narxi

Izolyatsiya yo'q: bitta oqim xatosi butun jarayonni o'ldiradi; **sinxronizatsiya** kerak.

	Jarayon	Oqim
Manzil maydoni	O'ziniki	Umumiy
Yaratish	Qimmat	Arzon
Almashtirish	Qimmatroq	Arzonroq
Ma'lumot almashish	IPC	To'g'ridan-to'g'ri
Xatoda	Boshqalarga ta'sir yo'q	Butun jarayon o'ladi
Misol	Chrome yorliqlari	Veb-server oqimlari

Oqimlarni amalga oshirish modellari



	N:1	1:1	M:N
+	Juda tez almashtirish	Haqiqiy parallellik	Afzalliklarni birlashtiradi
–	Bloklovchi chaqiruv hammasini to'xtatadi; parallellik yo'q	Yaratish qimmatroq	Murakkab
Misol	Eski Java green threads	Linux NPTL, Windows	Go, Java 21 virtual

Linuxda oqimlar va pthreads

Linuxda jarayon ham, oqim ham — **task**. Farq — `clone()` bayroqlarida:

```
fork() ~ clone(SIGCHLD)
pthread_create() ~ clone(CLONE_VM |
    CLONE_FILES | CLONE_THREAD | ...)
```

Oqimlarda PID (TGID) bir xil, TID har xil.

```
ps -elf | head          # LWP = TID
ls /proc/<PID>/task
top -H -p <PID>
```

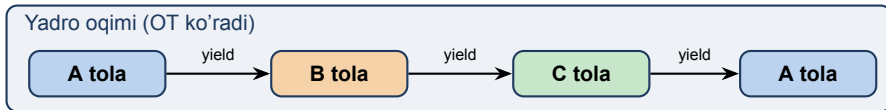
```
void *worker(void *arg) {
    int id = *(int *)arg;
    printf("worker %d started\n", id);
    return NULL;
}

int main(void) {
    pthread_t t[3]; int ids[3] = {1,2,3};
    for (int i = 0; i < 3; i++)
        pthread_create(&t[i], NULL,
                       worker, &ids[i]);
    for (int i = 0; i < 3; i++)
        pthread_join(t[i], NULL);
}
```

Satrlar tartibi **aniqlanmagan** — uni rejalashtiruvchi tanlaydi.

8. Tolalar (fibers) va korutinalar

Tola: yengil, kooperativ "oqim"



	Oqim (thread)	Tola (fiber)
Kim rejalashtiradi	OT yadrosi	Dastur (kutubxona)
Turi	Siqib chiqaruvchi	Kooperativ
Almashtirish narxi	Tizim chaqiruvi, mikrosekund	Bir nechta registr, nanosekund
Parallellik	Ha	Bitta oqim ichida yo'q
Bloklovchi chaqiruv	Bitta oqimni	Shu oqimdagi barcha tolalarni
Soni	Minglab	Millionlab

Windows: ConvertThreadToFiber, CreateFiber, SwitchToFiber. POSIX: swapcontext (eskirgan).

Korutina — to'xtab, keyin o'sha joydan davom etadigan funksiya.

- `async/await`: Python, C#, JS, Rust
- Go gorutinalari (M:N)
- Java 21 virtual oqimlari

Qayerda foydali

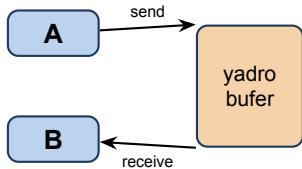
Tarmoqni kutadigan juda ko'p ulanishli serverlar: har ulanishga yadro oqimi — juda qimmat.

	Jarayon	Oqim	Tola
Xotira izolyatsiyasi	Ha	Yo'q	Yo'q
Kim rejalashtiradi	OT	OT	Dastur
Siqib chiqarish	Ha	Ha	Yo'q
Parallellik	Ha	Ha	Oqimlar orqali
Narx	Yuqori	O'rta	Past

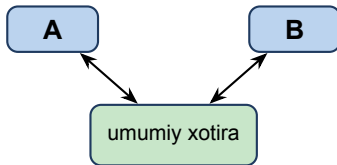
9. Jarayonlararo aloqa (IPC)

IPC ning ikki modeli

Xabar uzatish



Umumiy xotira



Afzalliklari

Kamchiliklari

Xabar uzatish

Oddiy, yadro sinxronlaydi, tarmoqda ham

Har xabar yadro orqali nusxalanadi

Umumiy xotira

Maksimal tezlik, nusxalash yo'q

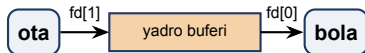
Sinxronizatsiya — dasturchi zimmasida

Mexanizm	Yo'nalish	Qarindoshlik	Tarmoq	Xususiyat
Anonim kanal	Bir tomonlama	Faqat qarindosh	Yo'q	Baytlar oqimi,
FIFO	Bir tomonlama	Istalgan	Yo'q	Fayl tizimida nom
Signallar	Xabarnoma	Istalgan	Yo'q	Faqat raqam
Xabarlar navbati	Ikki tomonlama	Istalgan	Yo'q	Chegaralar, ustuvorlik
Umumiy xotira	Umumiy	Istalgan	Yo'q	Eng tez, sinxronizatsiya kerak
Unix soketi	Ikki tomonlama	Bitta mashina	Yo'q	Deskriptor uzatish
TCP/UDP soketi	Ikki tomonlama	Istalgan	Ha	Universal
mmap fayl	Umumiy	Istalgan	Yo'q	Faylga bog'langan xotira

10. Kanallar (pipe, FIFO) va signallar

Anonim kanal (pipe)

```
int fd[2];          // 0: o'qish, 1: yozish
pipe(fd);
pid_t pid = fork();
if (pid == 0) {      // bola
    close(fd[1]);
    char buf[64];
    ssize_t n = read(fd[0], buf, 63);
    buf[n] = '\0';
    printf("child got: %s\n", buf);
    return 0;
}
close(fd[0]);        // ota
write(fd[1], "salom, bola!", 12);
close(fd[1]);        // -> EOF
wait(NULL);
```



- Bo'sh kanaldan read — **bloklanadi**
- To'la kanalga write — bloklanadi (Linux: 64 KiB)
- Barcha yozuvchilar yopilsa — read = 0 (**EOF**)
- O'quvchisiz yozish — SIGPIPE

Odatiy xato

Keraksiz uchni yopmaslik: bola fd[1] ni yopmasa, EOF ni hech qachon olmaydi va qotadi.

Nomlangan kanal (FIFO)

```
mkfifo /tmp/myfifo
ls -l /tmp/myfifo      # prw-r--r-- : 'p' turi
# 1-terminal:
cat /tmp/myfifo        # yozuvchini kutadi
# 2-terminal:
echo "salom" > /tmp/myfifo
rm /tmp/myfifo
```

Farqi

Anonim kanalni faqat **qarindosh** jarayonlar ishlatadi. FIFO ning fayl tizimida **nomi** bor — istalgan jarayon topadi.

Muhim

Ma'lumot **diskka yozilmaydi** — nom faqat kanalni topish uchun.

Har kuni: `ps aux | grep python | wc -l`

Signal	№	Standart	Qachon
SIGINT	2	Tugatish	Ctrl+C
SIGKILL	9	Tugatish	ushlab bo'lmaydi
SIGSEGV	11	Tugatish + dump	Noto'g'ri xotira
SIGPIPE	13	Tugatish	O'quvchisiz kanal
SIGTERM	15	Tugatish	Muloyim so'rov
SIGCHLD	17*	E'tiborsiz	Bola tugadi
SIGCONT	18*	Davom etish	fg, bg
SIGSTOP	19*	To'xtatish	ushlab bo'lmaydi
SIGTSTP	20*	To'xtatish	Ctrl+Z

* x86/ARM Linux raqamlari; skriptda nom ishlatish: `kill -TERM`

Signal — **asinxron xabarnoma**; faqat **raqam**, ma'lumot yo'q.

Jarayon tanlaydi: standart harakat, e'tiborsiz qoldirish yoki **ishlovchi**.

Avval TERM, keyin KILL

SIGTERM — saqlash, fayllarni yopish imkoni. SIGKILL — bir zumda, ma'lumot yo'qolishi mumkin.

```
volatile sig_atomic_t stop = 0;

void on_sigint(int sig) { stop = 1; }

int main(void) {
    struct sigaction sa = {0};
    sa.sa_handler = on_sigint;
    sigaction(SIGINT, &sa, NULL);
    while (!stop) pause();
    printf("SIGINT received\n");
}
```

Qoida

Ishlovchida faqat **async-signal-safe** funksiyalar. `printf`, `malloc` — **yo'q**. Faqat bayroq o'rnating.

```
kill -l
kill -TERM 4312
kill -9 4312          # oxirgi chora
kill -STOP 4312; kill -CONT 4312
trap 'echo ushlandi' INT
```

11. Umumiy xotira, navbatlar, sokerlar

```
// Yozuvchi
int fd = shm_open("/demo_shm",
                  O_CREAT | O_RDWR, 0600);
ftruncate(fd, 4096);
char *p = mmap(NULL, 4096,
               PROT_READ | PROT_WRITE,
               MAP_SHARED, fd, 0);
strcpy(p, "salom from shared memory");

// O'quvchi
int fd = shm_open("/demo_shm", O_RDONLY, 0);
char *p = mmap(NULL, 4096, PROT_READ,
               MAP_SHARED, fd, 0);
printf("read: %s\n", p);
shm_unlink("/demo_shm");
```

Bir xil jismoniy xotira ikkala jarayon manzil maydoniga akslanadi — **eng tez** IPC.

Sinxronizatsiya yo'q!

O'quvchi yarim yozilgan ma'lumotni ko'rishi mumkin. Semafor (`sem_open`) yoki process-shared myuteks bilan ishlating.

```
ls -l /dev/shm      # POSIX obyektlari
ipcs -a             # System V
                   obyektlari
```

Xabarlar navbati

Xabar **chegaralari** va **ustuvorliklar** saqlanadi (kanal — baytlar oqimi).

```
mqd_t q = mq_open("/demo_mq",  
                  O_CREAT | O_RDWR, 0600, NULL);  
mq_send(q, "task-42", 8, 5); // prio 5  
mq_receive(q, buf, sizeof buf, &prio);  
mq_unlink("/demo_mq");
```

Soketlar

Bir xil interfeys: socket, bind, listen, accept, connect, send, recv

- **mahalliy** — AF_UNIX (Docker, systemd)
- **tarmoq** — TCP/UDP

```
ss -xl | head      # Unix soketlari  
ss -tlnp           # TCP + jarayonlar
```

Yagona

Faqat soketlar **turli kompyuterlar** orasida ishlaydi.

Windows	Linuxdagi analogi
Anonim kanallar (CreatePipe)	pipe()
Nomlangan kanallar (\\.\pipe\name)	FIFO, lekin ikki tomonlama va tarmoqda
CreateFileMapping + MapViewOfFile	shm_open + mmap
Mailslots	Keng tarqatiladigan datagrammalar
Winsock	BSD soketlari
COM / RPC / ALPC	D-Bus, RPC
Signallar yo'q; hodisalar (events)	Signallar

12. Poyga holati va kritik seksiya

Poyga holati (race condition)

```
#define N 1000000
long counter = 0;           // umumiy

void *worker(void *arg) {
    for (int i = 0; i < N; i++)
        counter++;          // atomar EMAS!
    return NULL;
}
// main: 2 ta oqim, keyin join
```

Haqiqiy natija:

```
counter = 1068612 (expected 2000000)
counter = 1049141 (expected 2000000)
```

Ta'rif

Poyga holati — natija amallar bajarilish tartibiga bog'liq bo'lgan vaziyat.

counter++ = **3 qadam**:

1. LOAD R ← counter
2. ADD R ← R + 1
3. STORE counter ← R

Har safar **boshqa** natija, kutilganidan kam.

Yo'qolgan yangilanish (lost update)

Qadam	A oqim	B oqim	counter
1	LOAD R=5		5
2		LOAD R=5	5
3	ADD R=6		5
4		ADD R=6	5
5	STORE 6		6
6		STORE 6	6 (7 bo'lishi kerak)

Data race

Ikki oqim bitta katakka sinxronizatsiyasiz murojaat qiladi va kamida biri yozadi. C/C++ da bu — **aniqlanmagan xatti-harakat**.

kirish (lock)
KRITIK SEKSIYA
chiqish (unlock)
qolgan kod

Seksiyada bir vaqtda **ko'pi bilan bitta** oqim.

1. **O'zaro istisno** — seksiyada ko'pi bilan bitta oqim
2. **Progress** — seksiya bo'sh bo'lsa, kim kirishi chekli vaqtda hal qilinadi
3. **Cheklangan kutish** — kutayotganda boshqalar cheksiz kira olmaydi (ochliksiz)

Oddiy bayroq ishlamaydi

`while (busy) {} busy = 1;` — tekshirish va o'rnatish **ikki amal**; ikkala oqim ham kirishi mumkin.

Peterson (2 oqim)

```
int flag[2] = {0, 0};
int turn;
void lock(int i) {
    int j = 1 - i;
    flag[i] = 1;    // kirmoqchiman
    turn = j;        // navbatni beraman
    while (flag[j] && turn == j) { }
}
void unlock(int i) { flag[i] = 0; }
```

3 talabni bajaradi, lekin zamonaviy CPU **qayta tartiblaydi** — xotira to'siqlari kerak.

Apparat yordami

- **Test-and-Set** — atomar o'qish va 1 yozish
- **Compare-and-Swap** — “expected bo'lsa, new yoz”
- **Fetch-and-Add** — x86: lock xadd

```
#include <stdatomic.h>
atomic_long counter = 0;
atomic_fetch_add(&counter, 1);

atomic_flag lk = ATOMIC_FLAG_INIT;
while (atomic_flag_test_and_set(&lk)) {}
atomic_flag_clear(&lk);
```

13. Sinxronizatsiya primitivlari

Myuteks va spin-blokirovka

```
pthread_mutex_t m =  
    PTHREAD_MUTEX_INITIALIZER;  
for (int i = 0; i < N; i++) {  
    pthread_mutex_lock(&m);  
    counter++;  
    pthread_mutex_unlock(&m);  
}  
// natija har doim 2000000
```

Myuteks — **egasi** bor: faqat bloklagan oqim ochadi. Band bo'lsa — oqim **uxlaydi**.

	Myuteks	Spinlock
Kutish	Uyqu	Faol (CPU sarflaydi)
Narx	Kontekst almashtirish	CPU sikllari
Qachon	Uzun seksiya, I/O	Juda qisqa, ko'p yadro
Qayerda	Dasturlar	Yadro, uzilishlar

Bir yadroda

Spinlock ma'nosiz: aylanayotganda egasi ishlay olmaydi va bo'shata olmaydi.

Semafor (Deykstra, 1965)

S — butun hisoblagich, 2 atomar amal:

- **wait(S)** (P): $S > 0$ bo'lsa $S--$, aks holda bloklanish
- **signal(S)** (V): $S++$, kutayotgan birini uyg'otish

```
sem_t slots;  
sem_init(&slots, 0, 3);    // 3 ta joy  
sem_wait(&slots);  
use_database();  
sem_post(&slots);
```

Tartiblash uchun:

```
sem_t done = 0;  
A:  step1();  sem_post(&done);  
B:  sem_wait(&done);  step2();
```

	Myuteks	Semafor
Qiymat	band / bo'sh	butun ≥ 0
Egasi	bor	yo'q
Vazifa	o'zaro istisno	sanash, signal

macOS da `sem_init` yo'q — `sem_open` ishlatilg.

```
pthread_mutex_lock(&m);
while (!condition)           // while!
    pthread_cond_wait(&cv, &m);
/* shart bajarildi, m egallangan */
pthread_mutex_unlock(&m);

// boshqa oqim:
pthread_mutex_lock(&m);
condition = 1;
pthread_cond_signal(&cv);
pthread_mutex_unlock(&m);
```

cond_wait atomar: **m ni qo'yib uxlaydi**, uyg'onganda m ni qayta oladi.

Nega while?

Soxta uyg'onishlar; boshqa oqim shartni “olib ketgan” bo'lishi mumkin.

- **Monitor** — ma'lumot + myuteks + shartlar (Java synchronized)
- **RW lock** — ko'p o'quvchi *yoki* bitta yozuvchi
- **To'siq** — N oqim bir-birini kutadi

Primitivni tanlash. Windows analoglari

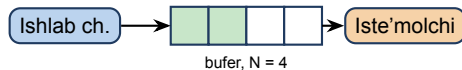
Vazifa	Primitiv	Windows	POSIX
Oddiy hisoblagich/bayroq	Atomar	CRITICAL_SECTION, SRWLOCK	mutex, rwlock
Ma'lumotlar tuzilmasi	Myuteks		
Ko'p o'qish, kam yozish	RW lock	CreateMutex (jarayonlararo)	process-shared mutex
Foydalanuvchilarni cheklash	Sanovchi semafor	CreateSemaphore	sem_t
Shartni kutish	Shartli o'zg. + myuteks	CreateEvent	cond var + bayroq
Jarayonlarni sinxronlash	Nomlangan semafor	Interlocked*	stdatomic.h
Yadroda qisqa seksiya	Spinlock	WaitForSingleObject	sem_wait, join

14. Sinxronizatsiyaning klassik masalalari

Ishlab chiqaruvchi–iste'molchi

```
semaphore mutex = 1; // o'zaro istisno
semaphore empty = N; // bo'sh joylar
semaphore full = 0; // band joylar
```

Ishlab chiqaruvchi:	Iste'molchi:
<code>wait(empty);</code>	<code>wait(full);</code>
<code>wait(mutex);</code>	<code>wait(mutex);</code>
<code>put(item);</code>	<code>item = take();</code>
<code>signal(mutex);</code>	<code>signal(mutex);</code>
<code>signal(full);</code>	<code>signal(empty);</code>



Tartib muhim!

`wait(mutex)` ni `wait(empty)` dan **oldin** qilsa va bufer to'la bo'lsa — mutex ushlangan holda uxlaydi: **o'zaro bloklanish**.

Hayotda: cmd1 | cmd2, oqimlar hovuzi navbati, chop etish buferi.

```
semaphore rw_mutex = 1, mutex = 1;  
int read_count = 0;
```

Yozuvchi:	O'quvchi:
wait(rw_mutex);	wait(mutex);
write();	read_count++;
signal(rw_mutex);	if (read_count == 1)
	wait(rw_mutex);
	signal(mutex);
	read();
	wait(mutex);
	read_count--;
	if (read_count == 0)
	signal(rw_mutex);
	signal(mutex);

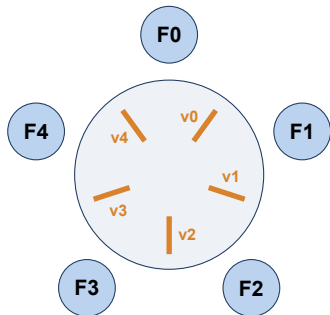
Qoida

O'quvchilar — **bir vaqtda**; yozuvchi — **yolg'iz**.

Birinchi o'quvchi yozuvchilarni bloklaydi, oxirgisi kiritadi.

Ochlik

O'quvchilar uzluksiz kelsa, yozuvchi abadiy kutadi. Tayyor yechim: `pthread_rwlock_t`.



5 faylasuf, 5 vilka. Ovqatlanish uchun **ikkala** qo'shni vilka kerak.

Sodda yechim

“Avval chap, keyin o'ng” — hammasi chapni olsa, **o'zaro bloklanish**.

Yechimlar:

1. **Tartiblash**: avval kichik raqamli vilka (F4: v0, keyin v4)
2. Stolda ko'pi bilan **4** faylasuf (semafor = 4)
3. **Asimmetriya**: juftlar chapdan, toqlar o'ngdan
4. Ikkala vilkani atomar olish

15. O'zaro bloklanish (deadlock)

O'zaro bloklanish va Koffman shartlari

```
// 1-oqim      // 2-oqim
lock(A);       lock(B);
lock(B);       lock(A);
// B ni kutadi // A ni kutadi
```

Hayotda

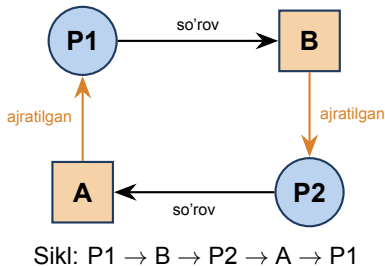
Tor ko'priksda ikki mashina: hech biri orqaga qaytmaydi.

Koffman (1971) — to'rttasi **bir vaqtda**:

- | | | |
|---|-------------------------|--|
| 1 | O'zaro istisno | Resursdan bitta jarayon foydalanadi |
| 2 | Ushlab turish va kutish | Ushlaydi va boshqasini kutadi |
| 3 | Siqib chiqarish yo'q | Faqat ixtiyoriy beriladi |
| 4 | Siklik kutish | $P1 \rightarrow P2 \rightarrow \dots \rightarrow P1$ |

Bittasini buzish kifoya!

Resurslarni taqsimlash grafi



- Doira — jarayon, kvadrat — resurs
- $P \rightarrow R$ — so'rov
- $R \rightarrow P$ — ajratilgan

Qoida

Har resursda **bitta nusxa**: sikl $\Leftrightarrow$ o'zaro bloklanish.

Bir nechta nusxa: sikl — zarur, lekin **yetarli emas**.

Yondashuv	G'oya	Misol
Oldini olish	4 shartdan birini buzish	Yagona egallash tartibi ; hammasini birdan so'rash
Chetlab o'tish	Faqat xavfsiz holatga olib kelsa berish	Bankir algoritmi
Aniqlash va tiklash	Sikllarni qidirish, bartaraf etish	Qurbonni tugatish, tranzaksiyani qaytarish (MBBT)
E'tiborsiz qoldirish	"Tuyaqush algoritmi"	Ko'pchilik umumiy OTlar

Dasturchi uchun eng amaliy

Barcha oqimlar har doim A ni B dan oldin egallasa — sikl hosil bo'la olmaydi.

Bankir algoritmi: holat

	Allocation	Max	Need
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3
P1	2 0 0	3 2 2	1 2 2
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1

Available = (3, 3, 2); jami (10, 5, 7)

G'oya

OT har jarayonning **maksimal ehtiyojini** biladi. Resurs faqat holat **xavfsiz** qolsa beriladi.

Need = Max – Allocation

Xavfsiz holat: hamma tugay oladigan tartib mavjud.

Bankir algoritmi: xavfsizlik tekshiruvi

Qadam	Jarayon	Need $\leq$ Work?	Work += Allocation
1	P1	$(1,2,2) \leq (3,3,2)$ ✓	(5,3,2)
2	P3	$(0,1,1) \leq (5,3,2)$ ✓	(7,4,3)
3	P0	$(7,4,3) \leq (7,4,3)$ ✓	(7,5,3)
4	P2	$(6,0,0) \leq (7,5,3)$ ✓	(10,5,5)
5	P4	$(4,3,1) \leq (10,5,5)$ ✓	(10,5,7)

Natija

Holat **xavfsiz**, ketma-ketlik: **⟨P1, P3, P0, P2, P4⟩**

So'rovda: Request $\leq$ Need; Request $\leq$ Available; shartli ajratib, tekshiruv; xavfli bo'lsa — qaytarish va kutish. **Xavfli** $\neq$ **bloklangan**: faqat kafolat yo'q.

Livelock

Oqimlar faol, lekin oldinga siljish yo'q. Yo'lakda bir-biriga bir vaqtda yo'l beradigan ikki odam.

Ochlik

Oqim resursni cheksiz ololmaydi — doim chetlab o'tiladi. Yechim: **qarish**, adolatli navbat.

Ustuvorlik inversiyasi

H kutadi L ning myuteksini, L ni M siqib chiqaradi. Yechim: **ustuvorlikni meros qilish**. Mars Pathfinder, 1997.

16. Linuxda kuzatish vositalari

Jarayonlarni kuzatish buyruqlari

Vazifa	Buyruq
Ro'yxat	<code>ps aux, ps -ef</code>
Daraxt	<code>pstree -p</code>
Real vaqt	<code>top, htop</code>
Oqimlar	<code>ps -eLf, top -H</code>
Topish	<code>pgrep -a, pidof</code>
Signal	<code>kill, pkill, killall</code>
Ustuvorlik	<code>nice, renice, chrt</code>
Tizim chaqiruvlari	<code>strace -f -p <PID></code>
Fayllar, kanallar	<code>ls -l -p, /proc/<PID>/fd</code>
IPC	<code>ipcs -a, /dev/shm, ss -xl</code>
Almashtirishlar	<code>vmstat 1, pidstat -w</code>

```
sleep 300 &  
jobs  
ps -o pid,ppid,stat,ni,cmd -p $!  
kill %1  
  
./pc &  
ps -o pid,nlwp,cmd -p $!  
# Linuxda myutekslar = futex  
strace -f -e trace=futex ./race
```

17. Amaliyot va xulosalar

1. **Jarayonlar daraxti:** `ps tree`, `sleep 100 &` ning PPID i, `/proc/<PID>/status`.
2. **fork va zombi:** `fork1.c`; wait siz zombi yaratish va topish.
3. **Jarayonlarni sanash:** sikl ichida `fork` va `fork()` || `fork()`.
4. **Poyga:** `race.c` ni 3 usulda tuzatish, vaqtni solishtirish.
5. **Kanal va FIFO:** yopilgan `fd[1]` ta'siri; FIFO orqali `wc -l`.
6. **Signallar:** INT, TERM, KILL farqi; `trap` li skript.
7. **Ishlab ch.–iste'molchi va nice:** `while` → `if`; bitta yadroda `nice 0` va `19`.

Muhit

Faqat Linux virtual mashinasi: `sudo apt install build-essential strace psmisc`

Xato	Oqibat	To'g'risi
wait() ni chaqirmaslik	Zombilar, PID tugaydi	waitpid yoki SIGCHLD
fork() oldidan to'la bufer	Chiqish takrorlanadi	fflush(stdout)
Kanal uchini yopmaslik	EOF yo'q, qotadi	Keraksiz uchni darhol yopish
Sinxronizatsiyasiz umumiy ma'lumot	Poyga	Myuteks, atomar
cond_wait oldidan if	Shart bajarilmagan	while
Blokirovkalar turli tartibda	O'zaro bloklanish	Yagona tartib
Kritik seksiyada I/O	Boshqalar bekor turadi	Seksiyani qisqartirish
Xato yo'lida unlock yo'q	Abadiy blokirovka	Yagona chiqish, RAI
Sinxronizatsiya o'rniga volatile	Poyga qoladi	atomic_*
Ishlovchida printf	libc ichida bloklanish	Faqat bayroq
Darhol kill -9	Ma'lumot yo'qoladi	Avval SIGTERM

1. Ko'p vazifalilik — CPU almashtirish illyuziyasi; siqish — **taymer**.
2. **Jarayon** — o'z manzil maydoni; hammasi **PCB** da.
3. Holatlar: **New, Ready, Running, Waiting, Terminated**.
4. **fork + exec**; **wait** bo'lmasa — zombi.
5. n ta `fork()` $\Rightarrow 2^n$ jarayon.
6. **SRTF** — min. kutish; **RR** — yaxshi javob.
7. **Oqim** — umumiy xotira, o'z stek va registrlari.
8. **Tola** — kooperativ, foydalanuvchi maydonida.
9. **IPC**: xabar uzatish yoki umumiy xotira.
10. `counter++` atomar emas $\Rightarrow$ **poyga**.
11. Kritik seksiya: **istisno, progress, cheklangan kutish**.
12. **Deadlock: yagona blokirovka tartibi!**

Nazorat savollari (1)

1. Konkurentlik parallellikdan nimasi bilan farq qiladi? Bitta yadroda konkurentlik mumkinmi?
2. Siqib chiqaruvchi ko'p vazifalilikni qaysi qurilma mumkin qiladi?
3. Dastur jarayondan nimasi bilan farq qiladi?
4. Nima uchun Waiting → Running o'tishi mumkin emas?
5. PCB da nima saqlanadi? Linuxda u qanday ataladi?
6. `fork()` ota va bola jarayonda nima qaytaradi?
7. Copy-on-write nima?
8. To'rtta ketma-ket `fork()` dan keyin nechta jarayon bo'ladi?
9. `exec()` nima qiladi? PID saqlanadimi?
10. Zombi va yetim jarayon nima? Zombini `kill -9` bilan o'ldirib bo'ladimi?
11. `CreateProcess` `fork` + `exec` dan nimasi bilan farq qiladi?
12. Kutish vaqti va aylanish vaqtiga ta'rif bering.
13. Karvon effekti nima? Qaysi algoritmgaga xos?
14. Nima uchun SJF ni amalda amalga oshirish qiyin?
15. Kvant o'lchami RR ga qanday ta'sir qiladi?

Nazorat savollari (2)

16. CFS dagi vruntime nima? Linux 6.6 dan qaysi rejalashtiruvchi?
17. Nima uchun oqimlar orasida almashtirish arzonroq?
18. Oqimlar nimani bo'lishadi, nima o'ziniki?
19. N:1, 1:1 va M:N modellarini solishtiring.
20. Tola oqimdan nimasi bilan farq qiladi?
21. "Xabar uzatish" va "umumiy xotira" modellarini solishtiring.
22. Nima uchun kanal o'quvchisi qotib qolishi mumkin?
23. SIGTERM va SIGKILL farqi? Qaysi signallarni ushlab bo'lmaydi?
24. Yangilanish yo'qoladigan counter++ navbatlashuvini ko'rsating.
25. Kritik seksiya yechimiga 3 talab.
26. Myuteks semafordan va spinlockdan nimasi bilan farq qiladi?
27. Nima uchun pthread_cond_wait oldidan while?
28. wait(empty) va wait(mutex) o'zni almashsa nima bo'ladi?
29. Koffman shartlari; yagona tartib qaysi birini buzadi?
30. 15-bo'limdagi holat xavfsizmi? Xavfsiz ketma-ketlikni toping.

1. Tanenbaum A., Bos H. *Modern Operating Systems*. 4th ed. — Pearson, 2014. — 2, 6-boblar.
2. Silberschatz A., Galvin P., Gagne G. *Operating System Concepts*. 10th ed. — Wiley, 2018. — 3–8-boblar.
3. Arpaci-Dusseau R., A. *Operating Systems: Three Easy Pieces*. — ostep.org (bepul).
4. Kerrisk M. *The Linux Programming Interface*. — No Starch Press, 2010.
5. Downey A. *The Little Book of Semaphores*. — Green Tea Press (bepul).
6. Russinovich M. va boshq. *Windows Internals*. 7th ed., Part 1. — 3–4-boblar.
7. man 2 fork, man 2 wait, man 7 pipe, man 7 signal, man 7 pthreads, man 7 sem_overview, man 7 sched.
8. Microsoft Learn: *Processes and Threads, Synchronization, Interprocess Communications*.

E'tiboringiz uchun rahmat!

Savollar?