Семинар 4

pthreads

IPC

BUG

BUG

BUG

BUG

BUG

BUG

BUG

BUG

BUG

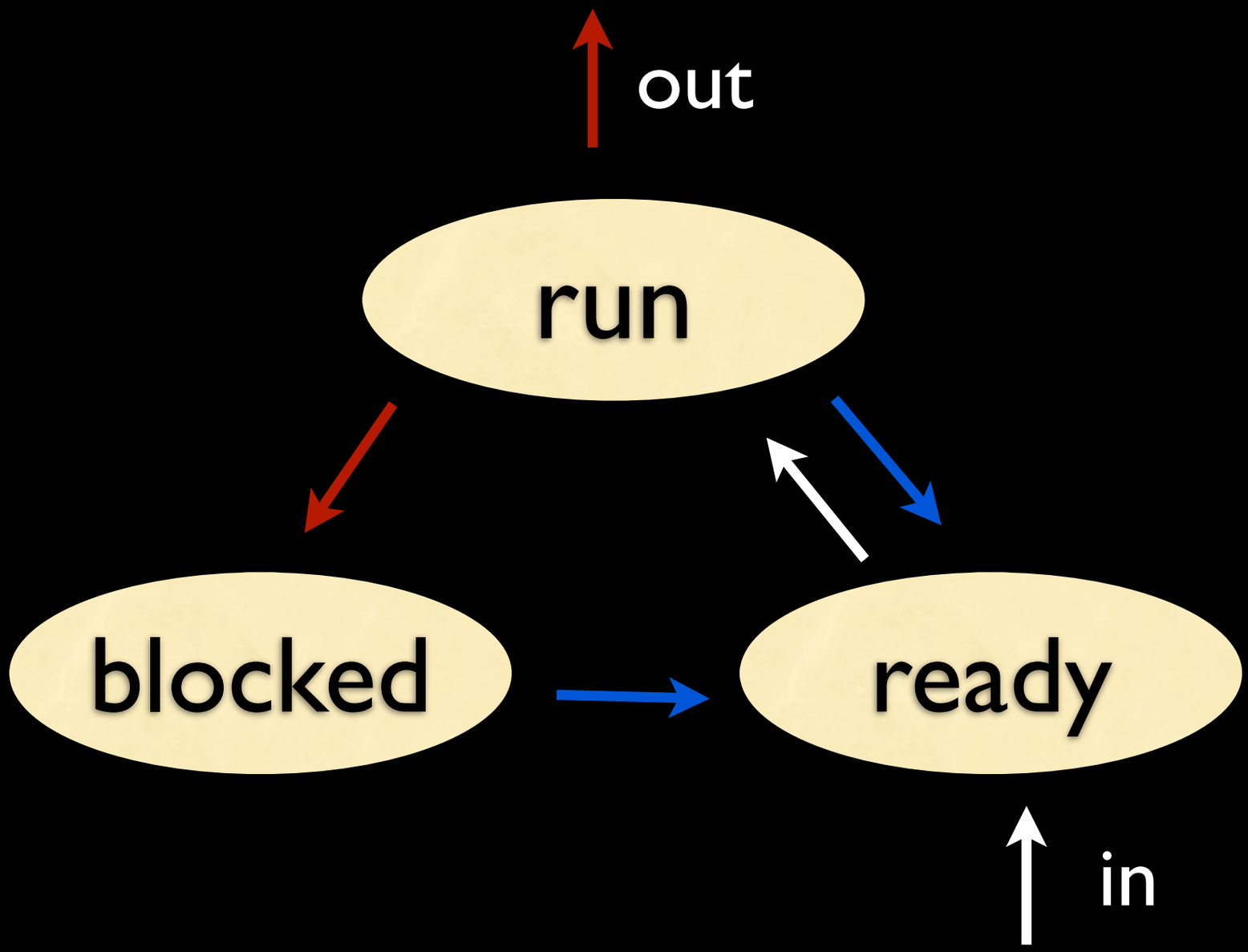
BUG

BUG

BUG

Зачем запускать домашний код?

Simple process states



Sheduling

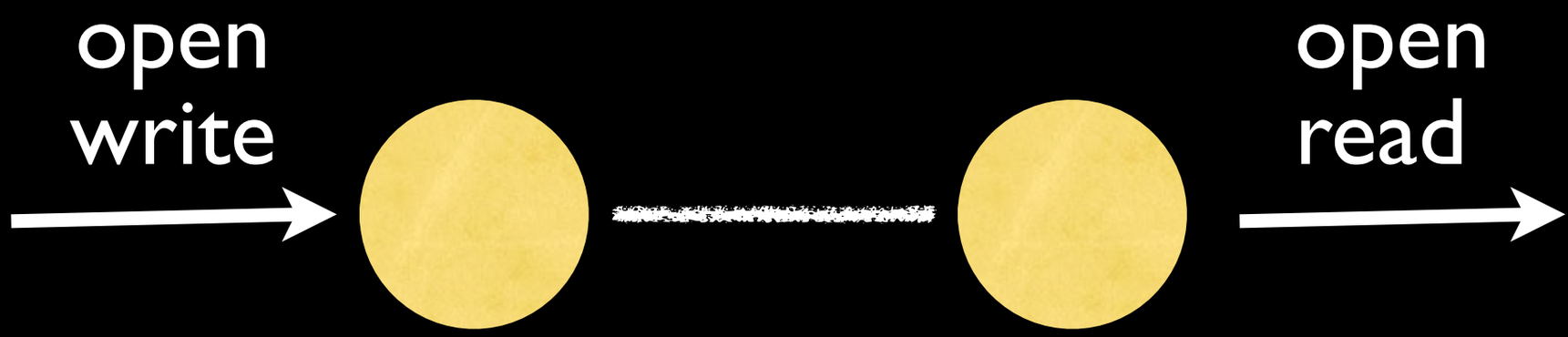
FCFS
Round Robin
Shortes Job First

pipes

s = pipe(fds)

named pipes (FIFO)

s = mknod(name, mode, dev)



threads

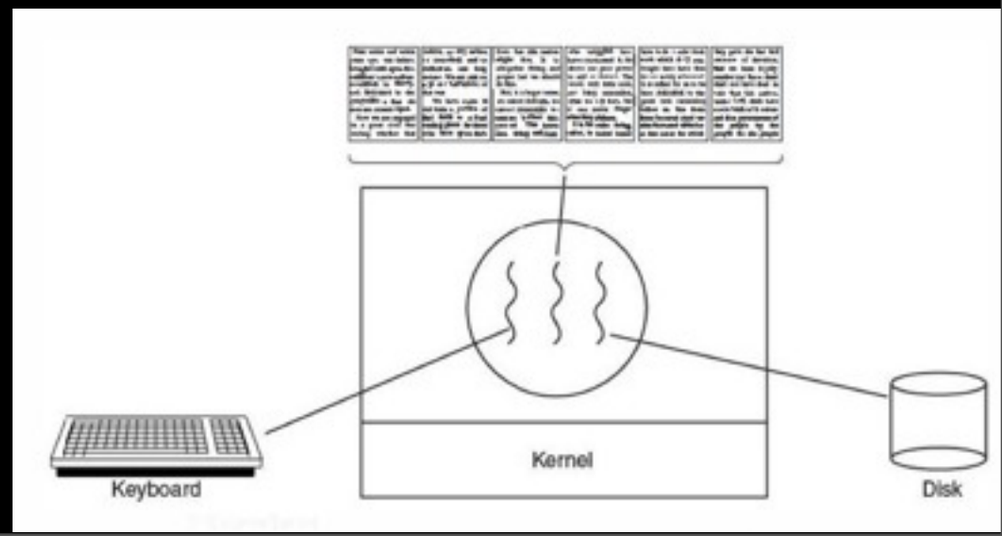
what for?

should be enherited while fork()?

- lightweight processes inside process
- 10 ÷100 faster creation
- shared process's stuff
- multithreading
- no protection between

what if one write, another close?

should be blocked together?



processes

address space

global variables

open files

child processes

pending alarm

signals & their handlers

account information

threads

program counter

registers

stack

state

threads

- thread_create(function_to_start_from)
- thread_exit
- thread_join(tid)
- thread_yield

pthreads

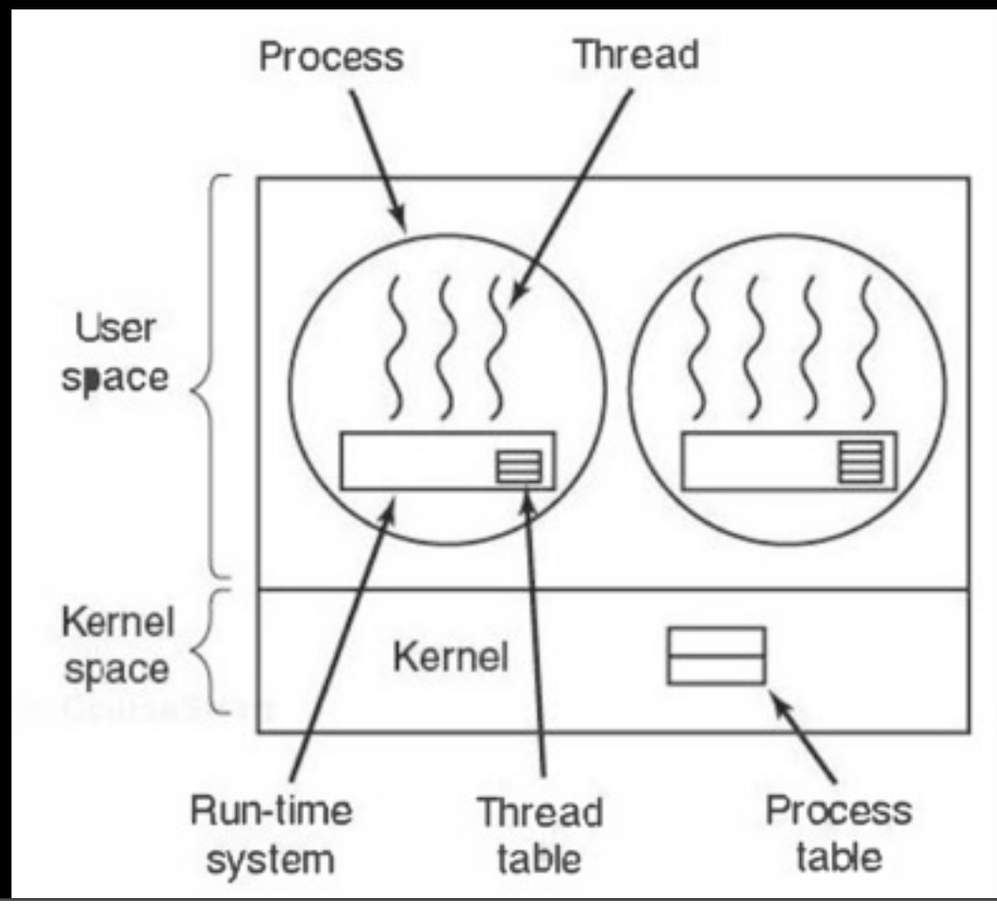
- tid
- registers (with program counter)
- attributes: stack size, scheduling params, ...

pthread_create, pthread_exit, pthread_join, pthread_yield

pthread_attr_init, pthread_attr_destroy

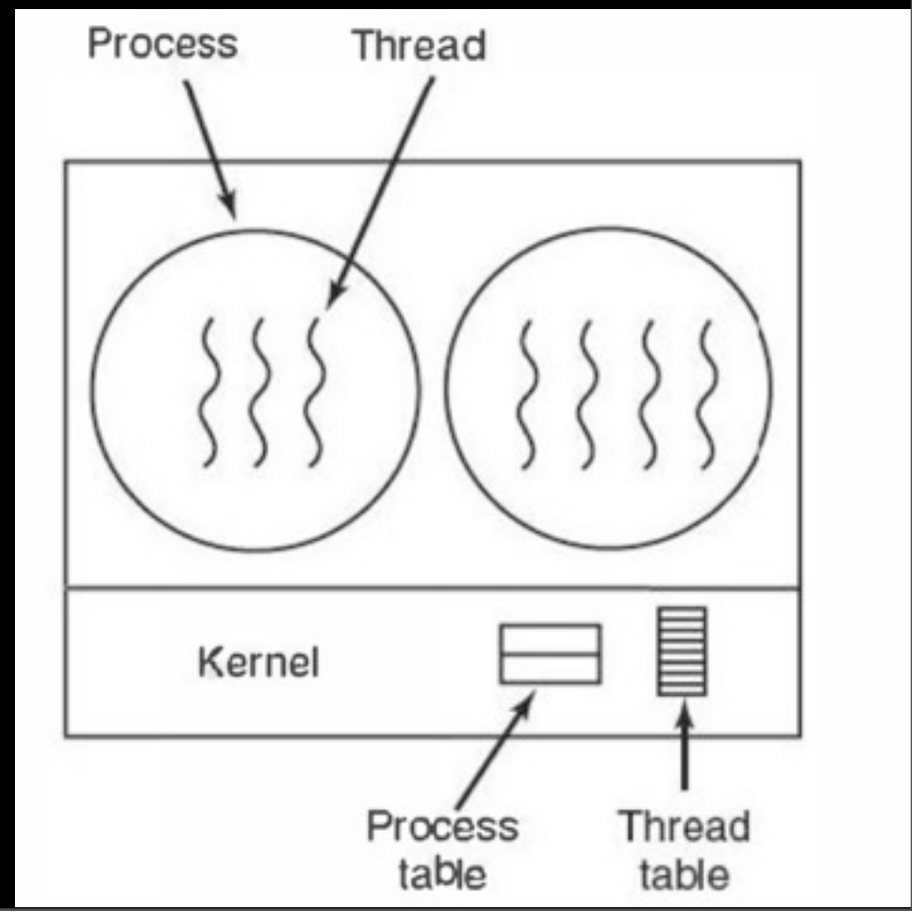
implementation in user-space

- could exists on OS with no support of threads
- one could block others by a syscall or page fault
- library implementation
- no context switches
- per process scheduler
- should wrap syscall (i.e.select)
- periodic clock interrupts



implementation in kernel-space

- thread table and state is in kernel
- no blocking by a syscall
- greater cost for creating and destroing
- threads recycle
- how much should be on fork?
- multiple signal handlers?



hybrid implementation

- each one has it's own

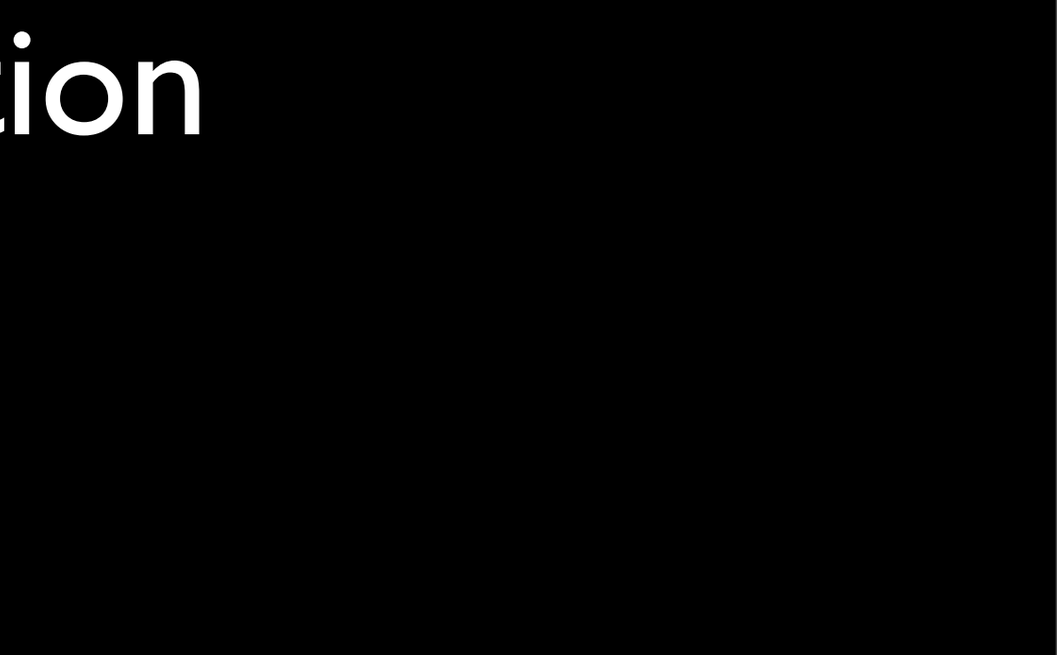


Figure 2-17. Multiplexing user-level threads onto kernel-level threads.

more threads

- pop-up threads
-