Chapter 3
Processes

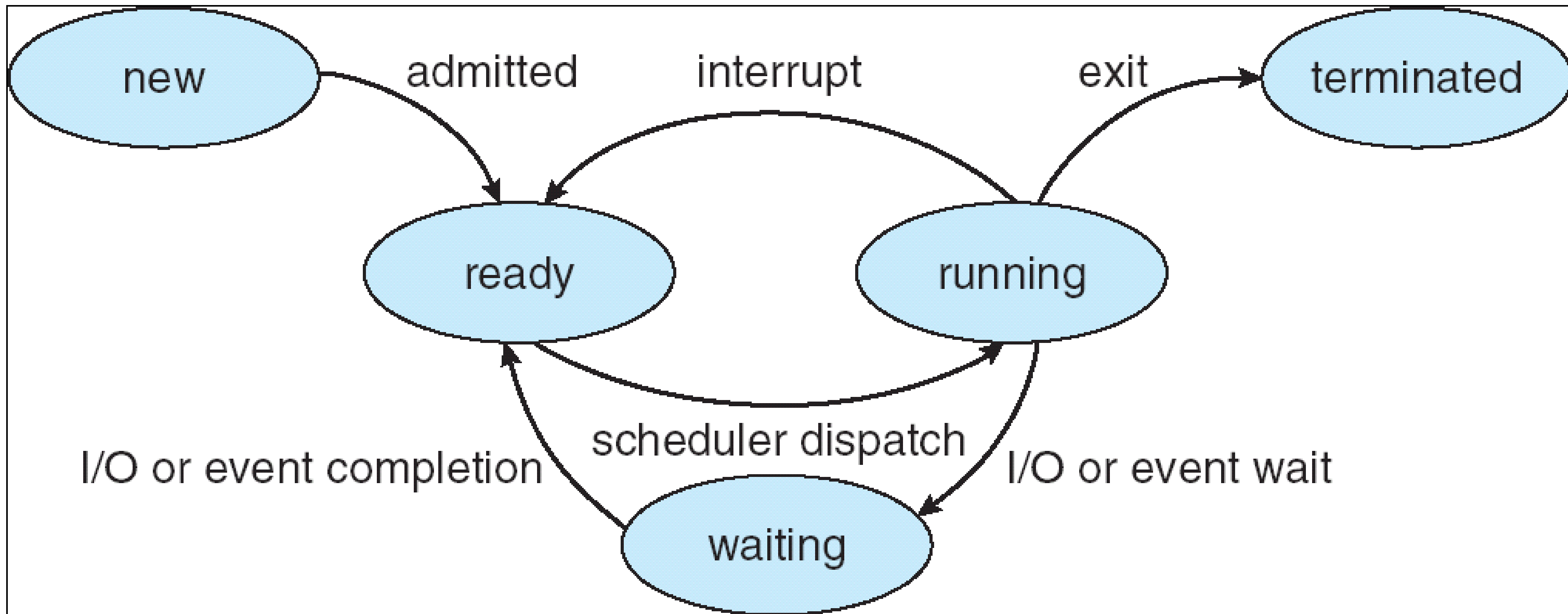
we will completely ignore threads today

Images from Silberschatz

Process

- Define:
- Memory Regions:
- Loaded from executable file:
  - ELF: Executable and Linkable Format
    - Linux
    - What does this contain?

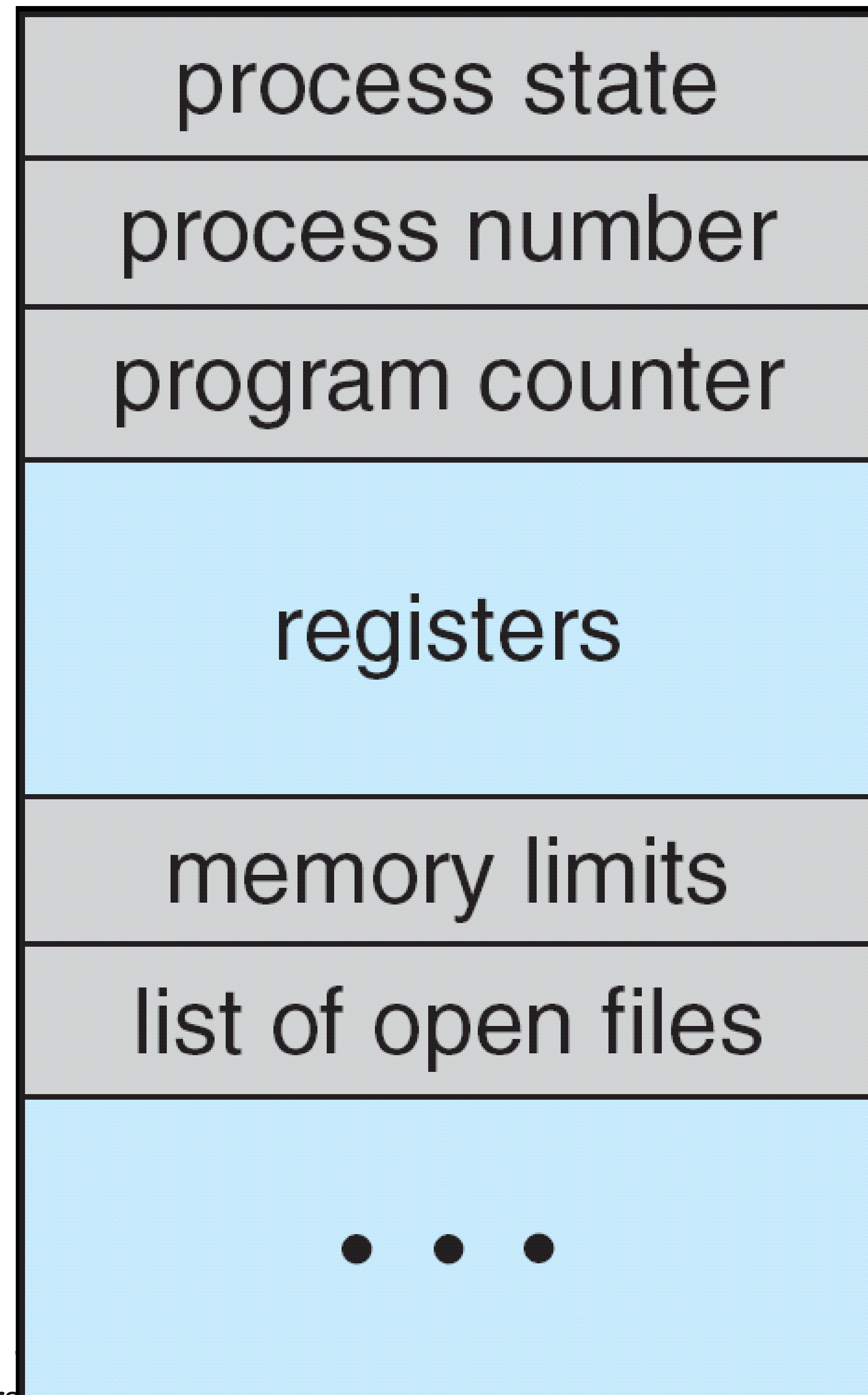
State Machine



- While a process is active it is in a particular state
- How many processes can be in each state?
- Data Structures? Where? Which kind? Why?

Process Control Block

- Who owns this data structure?
- CPU Scheduling data
- Memory Management data
- Accounting data



Types of Processes

- I/O Bound

- CPU Bound

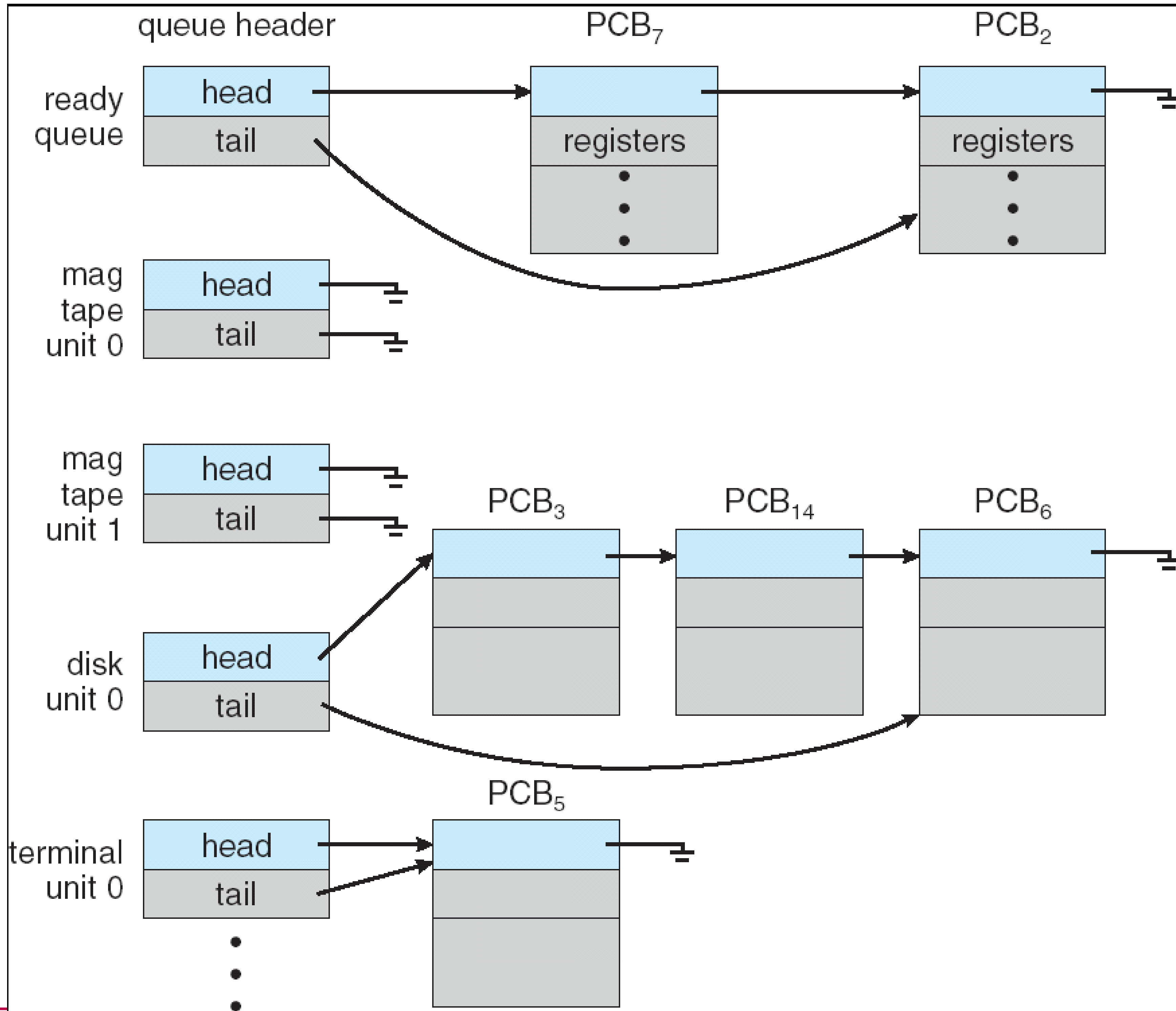
- How does this affect the OS?

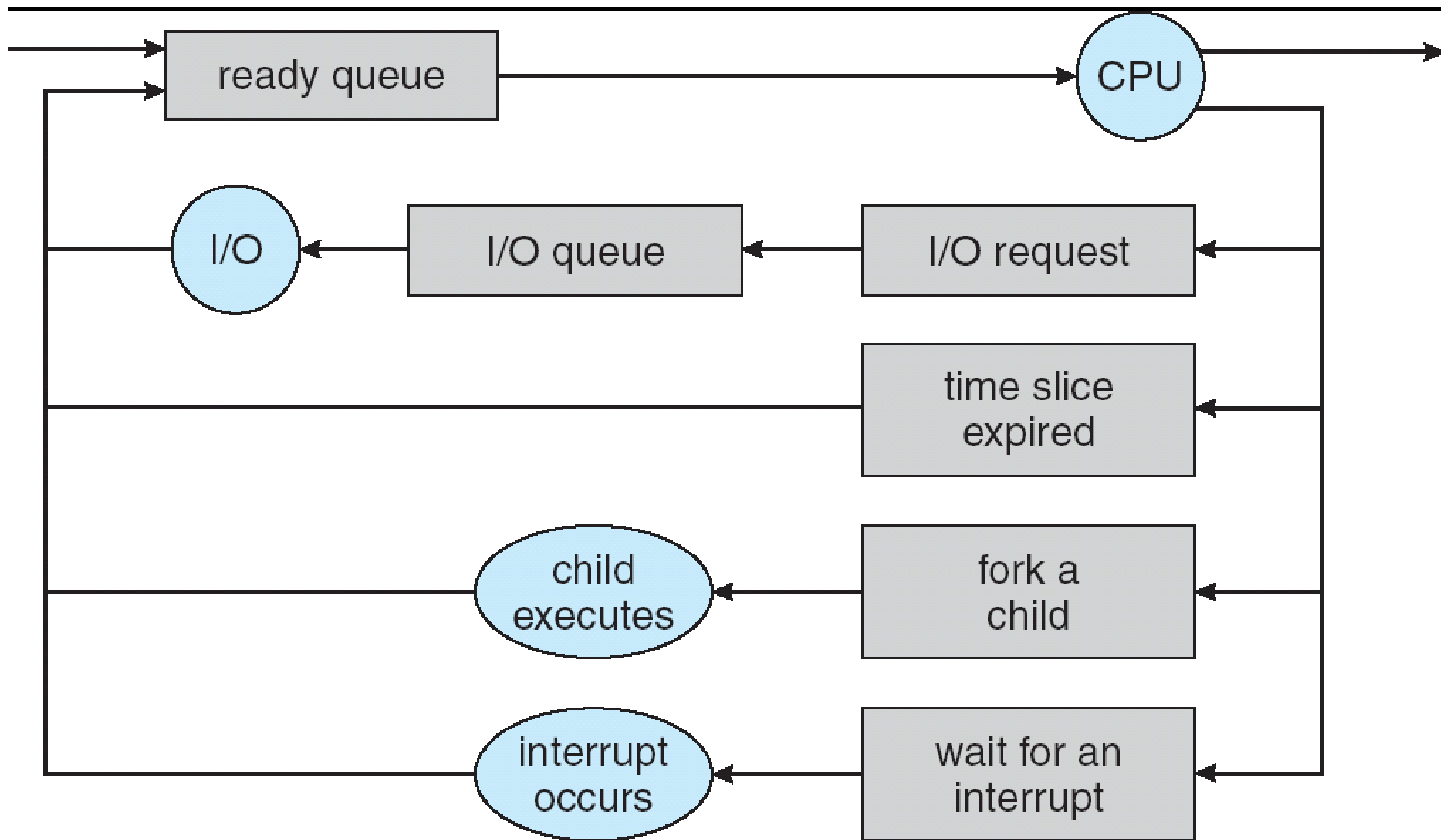
Process Scheduling

- Process Scheduler
  - Purpose:
  - Data structures:
  - Dispatched:

Schedulers

- Job Scheduler
  - Long term
  - Why is this important?
- CPU Scheduler
  - Short term
  - Constraints?
- Many OSes (Unix/Windows) don't really have a Job Scheduler


queue header
PCB7
PCB2
ready queue
head
tail
registers
registers
mag tape unit 0
head
tail
mag tape unit 1
head
tail
PCB3
PCB14
PCB6
disk unit 0
head
tail
PCB5
terminal unit 0
head
tail

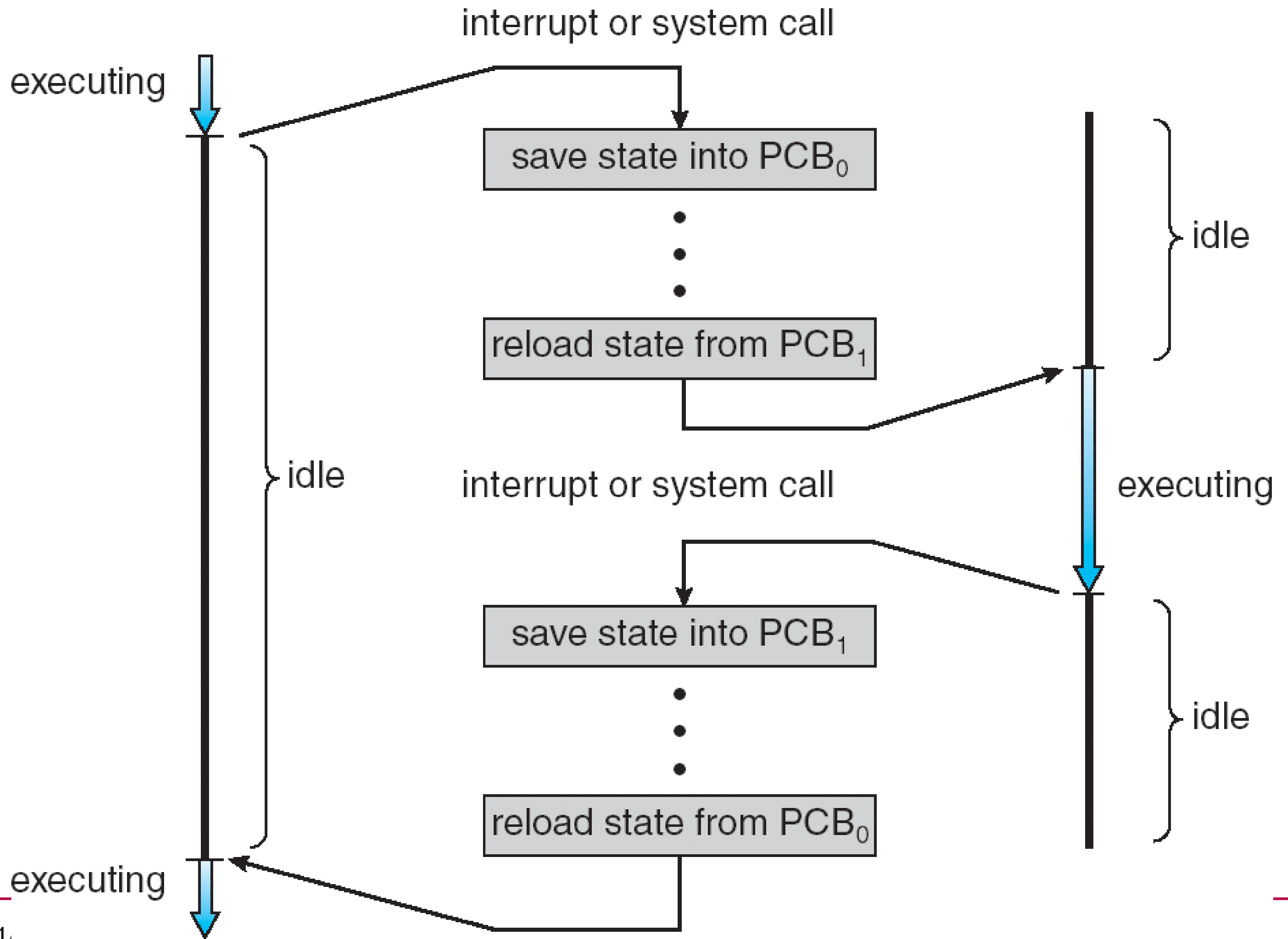

ready queue
CPU
I/O
I/O queue
I/O request
time slice expired
child executes
fork a child
interrupt occurs
wait for an interrupt

Context Switch

- Context:

- What happens during a Context Switch?

- Speed?



process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
idle
reload state from PCB_1
idle
interrupt or system call
executing
save state into PCB_1
idle
reload state from PCB_0
executing

Process Creation

```
/* This code works on Zeus! */
int main()
{
    pid_t  pid;
    int value = 0;
    value = 9;

    /* fork another process */
    pid = fork();
    fprintf(stderr,"The value: %d", value);

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait (NULL);
        printf ("Child Complete");
        exit(0);
    }
}   /* page 92 of Silberschatz */
```

What happens if we put an fprintf() in side the block after the execlp()?

Process Termination

- kill(pid, signal)

  $ man kill

  $ ps u

  $ kill -9 pid

  $ man -s 2 kill

  $ man -s 7 signal

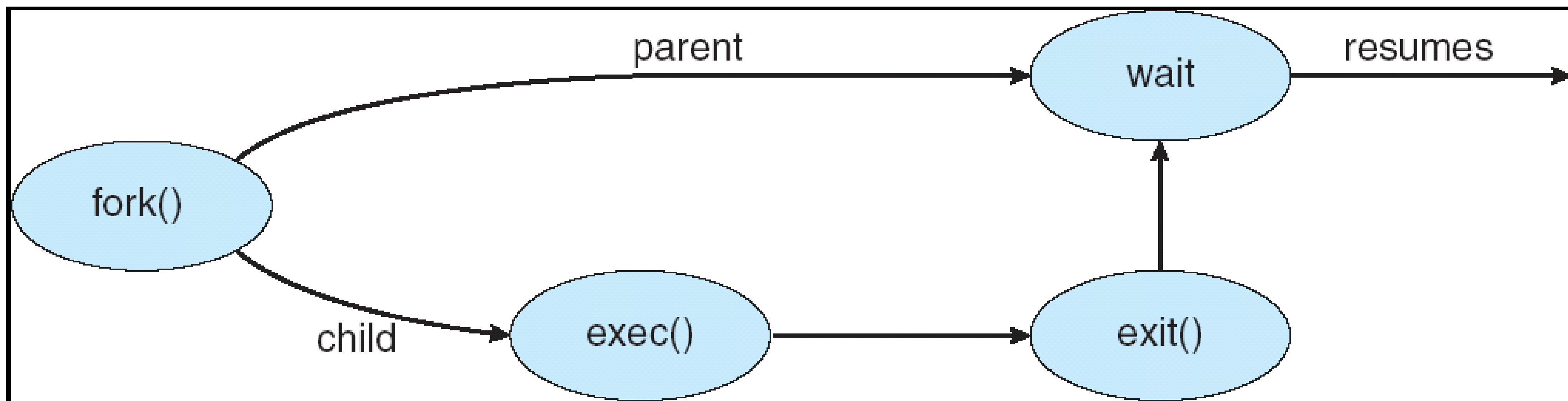
- Cascading termination:

Windows (Win32 API)

- CreateProcess()
  - fork() and exec() rolled into one
  - 10 parameters!

- WaitForSingleObject()

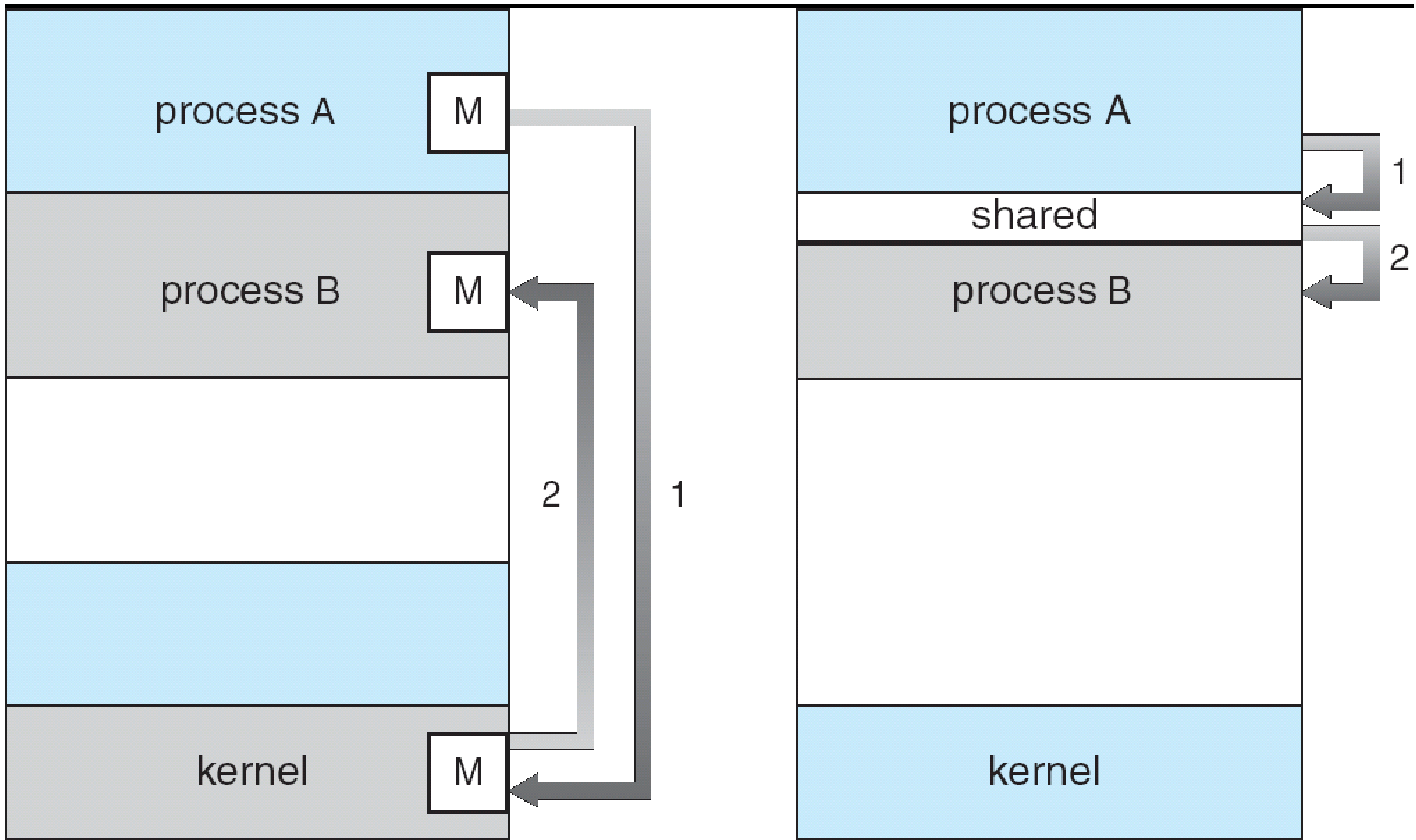
- TerminateProcess()


parent
wait
resumes
fork()
child
exec()
exit()

Interprocess Communication

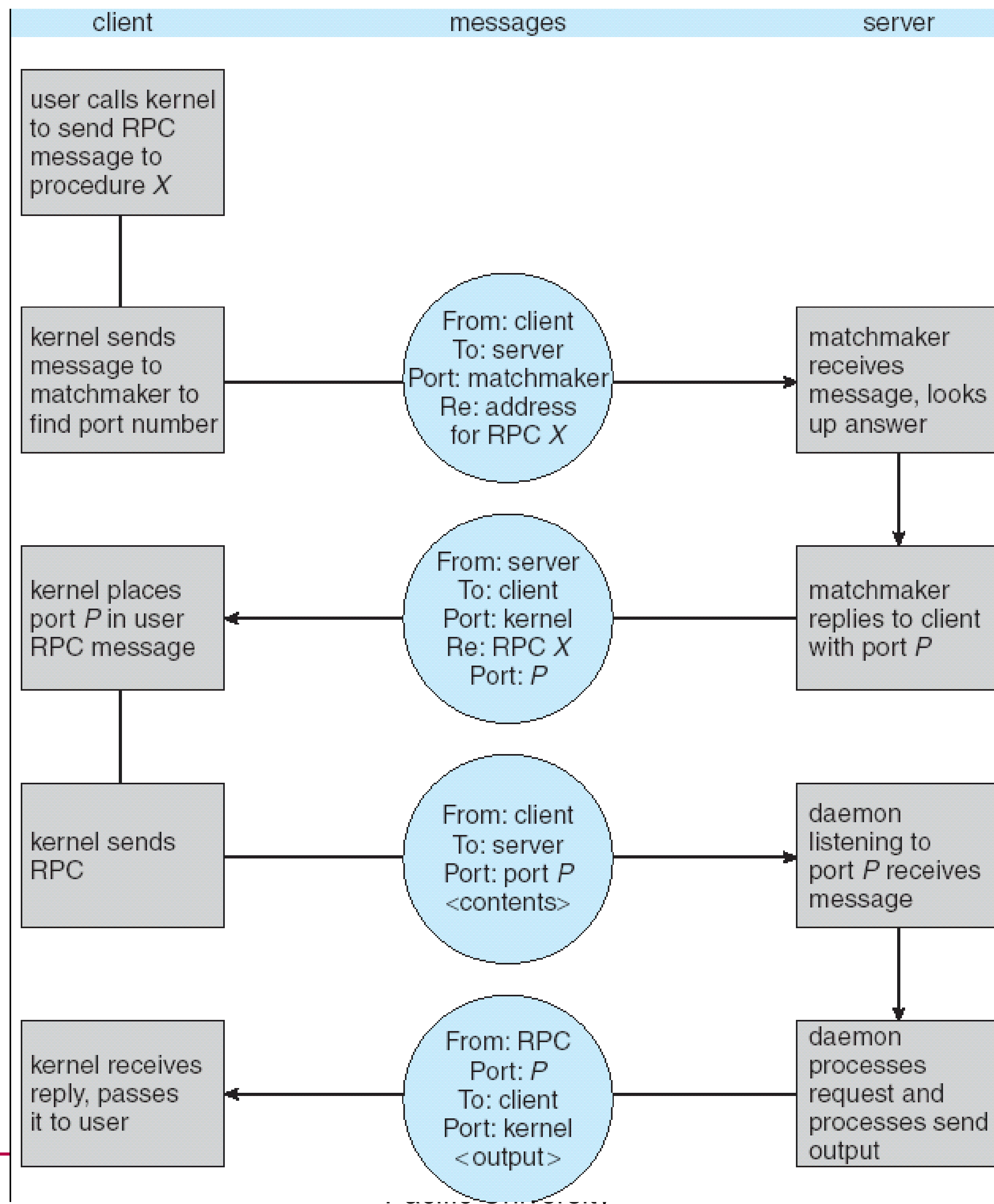
- Why do we want this?

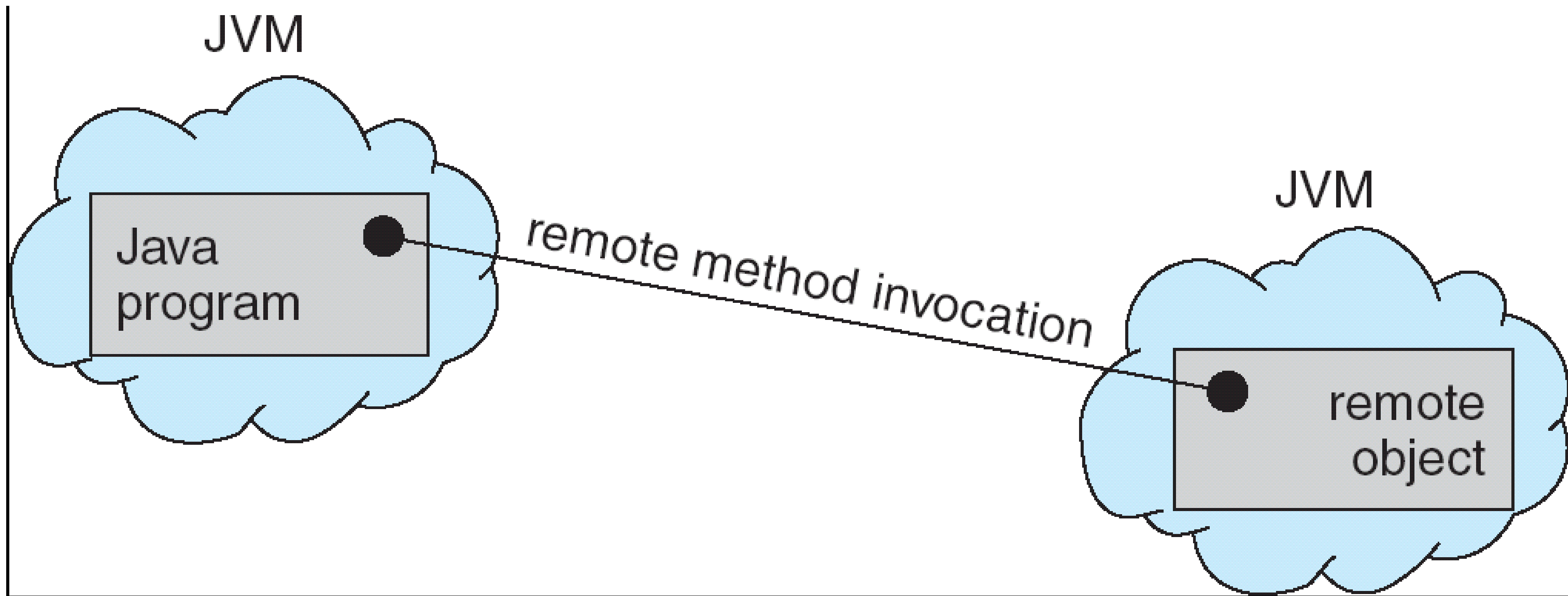
- Types:
  - Shared memory:
  - Message passing:


process A
M
process B
M
2
1
kernel
M
(a)
process A
1
shared
2
process B
kernel
(b)

Shared Memory

```
/* This code works on Zeus! */
int main()
{
   int segment_id;
   char *shared_memory;
   const int size = 4096;

   /* allocate shared memory segment */
   segment_id = shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);

   /* attach the shared memory segment */
   shared_memory = (char*) shmat (segment_id, NULL, 0);

   /* write a message to the shared memory segment */
   sprintf(shared_memory, "Hi there!");

   /* now print out the string from shared memory */
   printf("*%s\n", shared_memory);

   /* now detach the shared memory segment */
   shmdt(shared_memory);

   /* now remove the shared memory segment */
   shmctl(segment_id, IPC_RMID, NULL);
}
/* page 104 of Silberschatz */
```


client
messages
server
user calls kernel to send RPC message to procedure X
kernel sends message to matchmaker to find port number
From: client To: server Port: matchmaker Re: address for RPC X
matchmaker receives message, looks up answer
kernel places port P in user RPC message
From: server To: client Port: kernel Re: RPC X Port: P
matchmaker replies to client with port P
kernel sends RPC
From: client To: server Port: port P <contents>
daemon listening to port P receives message
kernel receives reply, passes it to user
From: RPC Port: P To: client Port: kernel <output>
daemon processes request and processes send output


JVM
Java program
remote method invocation
JVM
remote object

Functions

```
int execl(const char *path, const char *arg, ...)

int execlp(const char *file, const char *arg, ...)

int execle(const char *file, const char *arg, ...,
   char const* envp[])

int execv(const char *path, char *const argv[])

int execvp(const char *file, char *const argv[])

int dup2(int oldfd, int newfd)

int pipe(int filedes[2])

pid_t waitpid(pid_t pid, int *status, int options)

char* strtok_r(char *str, const char* delim, char **saveptr)
```

dup2()

```c
#define MAXLEN 1024
#include <unistd.h>

/* This code works on Zeus! */
int main()
{
   /* dup2(int oldfd, int newfd)  makes newfd be
    * the copy of oldfd closing newfd first if necessary.
    */

    char data[MAXLEN];
    int fd;

    fd = open("test.txt", O_WRONLY | O_CREAT | O_TRUNC, S_IRWXU);
    dup2(fd, STDOUT_FILENO);

    fprintf(stderr,"> ");
    fgets(&(data[0]),MAXLEN,stdin);

    write(fd, &(data[0]), strlen(data));

    printf("%s\n", data);

    close(fd);

    printf("ONCE MORE: %s\n", data);
}
```

dup2()

Let's investigate with gdb .....
gcc -g -o duptest duptest.c

```
chadd@coffee:~/test> gdb duptest
GNU gdb (GDB) SUSE (7.1-3.12)
(gdb) break dup2
Breakpoint 1 at 0x400690
(gdb) run
Breakpoint 1, 0x00007ffff7b41da0 in dup2 () from /lib64/libc.so.6
(gdb) disas 0x00007ffff7b41da0
Dump of assembler code for function dup2:
=> 0x00007ffff7b41da0 <+0>:     mov    $0x21,%eax
   0x00007ffff7b41da5 <+5>:     syscall
   0x00007ffff7b41da7 <+7>:     cmp    $0xfffffffffffff001,%rax
   0x00007ffff7b41dad <+13>:    jae    0x7ffff7b41db0 <dup2+16>
   0x00007ffff7b41daf <+15>:    retq
   ...
End of assembler dump.

(gdb) x 0x00007ffff7b41da5
0x7ffff7b41da5 <dup2+5>:        0x3d48050f
(gdb) up
(gdb) print/x &fd
$2 = 0x7fffffffdcec
```

What do these addresses tell us about process layout?

x86 Assembly

SYSCALL—Fast System Call

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 05	SYSCALL	NP	Valid	Invalid	Fast call to privilege level 0 system procedures.

0x21 = ?

```
#define __NR_dup                                32
__SYSCALL(__NR_dup, sys_dup)
#define __NR_dup2                               33
__SYSCALL(__NR_dup2, sys_dup2)
```

cat /proc/[pid]/maps

```
r = read
w = write
x = execute
s = shared
p = private (copy on write)
```

```
00400000-00401000 r-xp 00000000 fd:05 4085540        /home/chadd/test/duptest
00600000-00601000 r--p 00000000 fd:05 4085540        /home/chadd/test/duptest
00601000-00602000 rw-p 00001000 fd:05 4085540        /home/chadd/test/duptest

7fa833811000-7fa833968000 r-xp 00000000 fd:04 919578        /lib64/libc-2.11.2.so
7fa833968000-7fa833b67000 ---p 00157000 fd:04 919578        /lib64/libc-2.11.2.so
7fa833b67000-7fa833b6b000 r--p 00156000 fd:04 919578        /lib64/libc-2.11.2.so
7fa833b6b000-7fa833b6c000 rw-p 0015a000 fd:04 919578        /lib64/libc-2.11.2.so
7fa833b6c000-7fa833b71000 rw-p 00000000 00:00 0
7fa833b71000-7fa833b90000 r-xp 00000000 fd:04 924641        /lib64/ld-2.11.2.so
7fa833d58000-7fa833d5b000 rw-p 00000000 00:00 0
7fa833d8c000-7fa833d8f000 rw-p 00000000 00:00 0
7fa833d8f000-7fa833d90000 r--p 0001e000 fd:04 924641        /lib64/ld-2.11.2.so
7fa833d90000-7fa833d91000 rw-p 0001f000 fd:04 924641        /lib64/ld-2.11.2.so

7fa833d91000-7fa833d92000 rw-p 00000000 00:00 0
7fffd1139000-7fffd115a000 rw-p 00000000 00:00 0             [stack]
7fffd11ff000-7fffd1200000 r-xp 00000000 00:00 0             [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0     [vsyscall]
```

simple pipe()

```c
#define MAXLEN 1024
#define READ 0
#define WRITE 1

/* This code works on Zeus! */
int main()
{
    char dataPipeWrite[MAXLEN];
    char dataPipeRead[MAXLEN];
    int thePipe[2];
    pid_t childPid;

    memset(&(dataPipeWrite[0]), '\0', MAXLEN);
    memset(&(dataPipeRead[0]), '\0', MAXLEN);

    pipe(thePipe);
                                        /* get data from user */
    readFromCommandLine(dataPipeWrite, MAXLEN);

    write(thePipe[WRITE], &(dataPipeWrite[0]), strlen(dataPipeWrite));
    read(thePipe[READ], &(dataPipeRead[0]), MAXLEN);

    fprintf(stderr,"READ FROM PIPE: %s\n",&(dataPipeRead[0]));

    close(thePipe[WRITE]);
    close(thePipe[READ]);
}
```

pipe()

```
void readFromCommandLine(char * data, int maxSize)
{
    fprintf(stderr,"> ");
    fgets(data, maxSize, stdin);
}

/* This code works on Zeus! */
int main()
{
  /* pipe(int filedes[2])  creates  a  pair of file
   * descriptors, pointing to a pipe inode, and places
   * them in the array pointed to by filedes.
   * filedes[0] is for reading,
   * filedes[1] is for writing.
   */

    char data[MAXLEN];
    int thePipe[2];
    pid_t childPid;

    memset(&(data[0]), '\0', MAXLEN);

    pipe(thePipe);

    childPid = fork();
```

pipe()

```
if(childPid == 0)
{
    /* I AM A CHILD */
    close(thePipe[WRITE]);
    read(thePipe[READ], &(data[0]), MAXLEN);

    while(strncmp( &(data[0]), "STOP", 4) != 0 )
    {
        printf("CHILD> %s\n", &(data[0]));
        read(thePipe[READ], &(data[0]), MAXLEN);
    }
    close(thePipe[READ]);
}
else
{
    close(thePipe[READ]);

    readFromCommandLine(&(data[0]), MAXLEN);
    write(thePipe[WRITE], &(data[0]), strlen(data));

    while(strncmp(&(data[0]), "STOP", 4) != 0)
    {
        readFromCommandLine(&(data[0]), MAXLEN);
        write(thePipe[WRITE], &(data[0]), strlen(data));
    }
    close(thePipe[WRITE]);
}
```