Chapter 3
Processes
we will completely ignore threads today

Images from Silberschatz

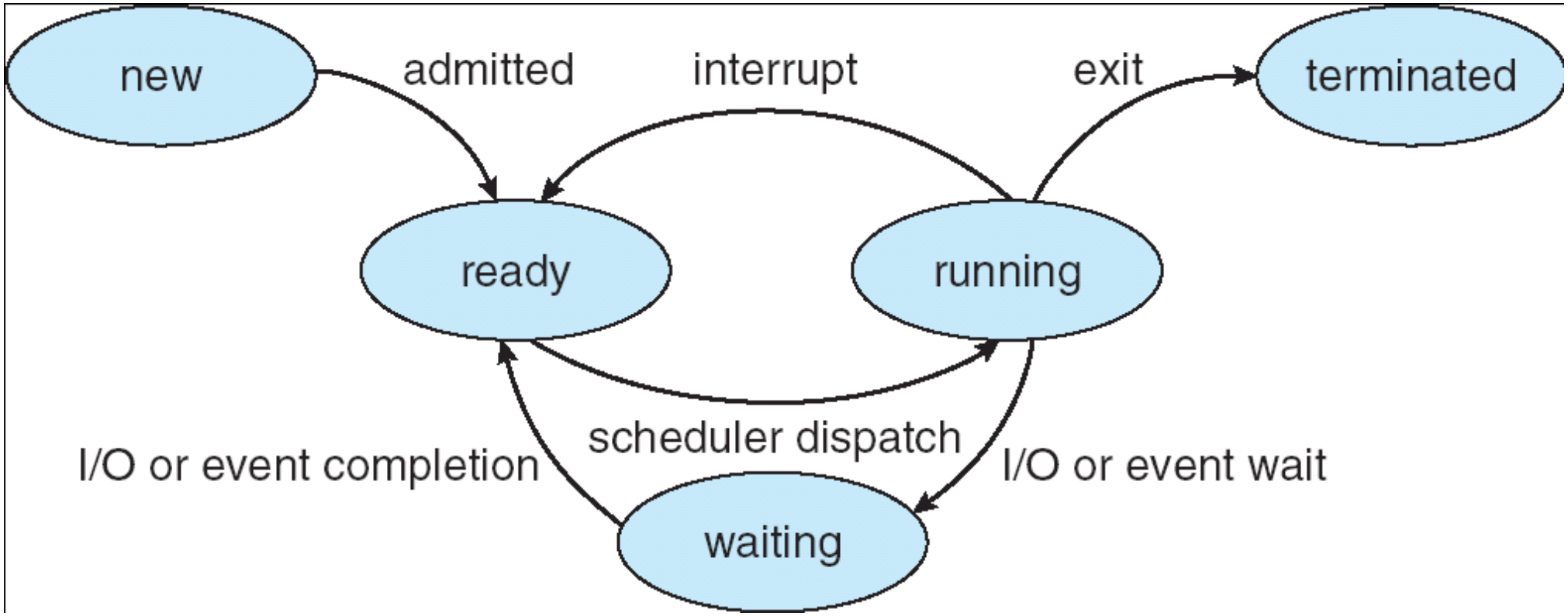
Process

- Define:

- Memory Regions:

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

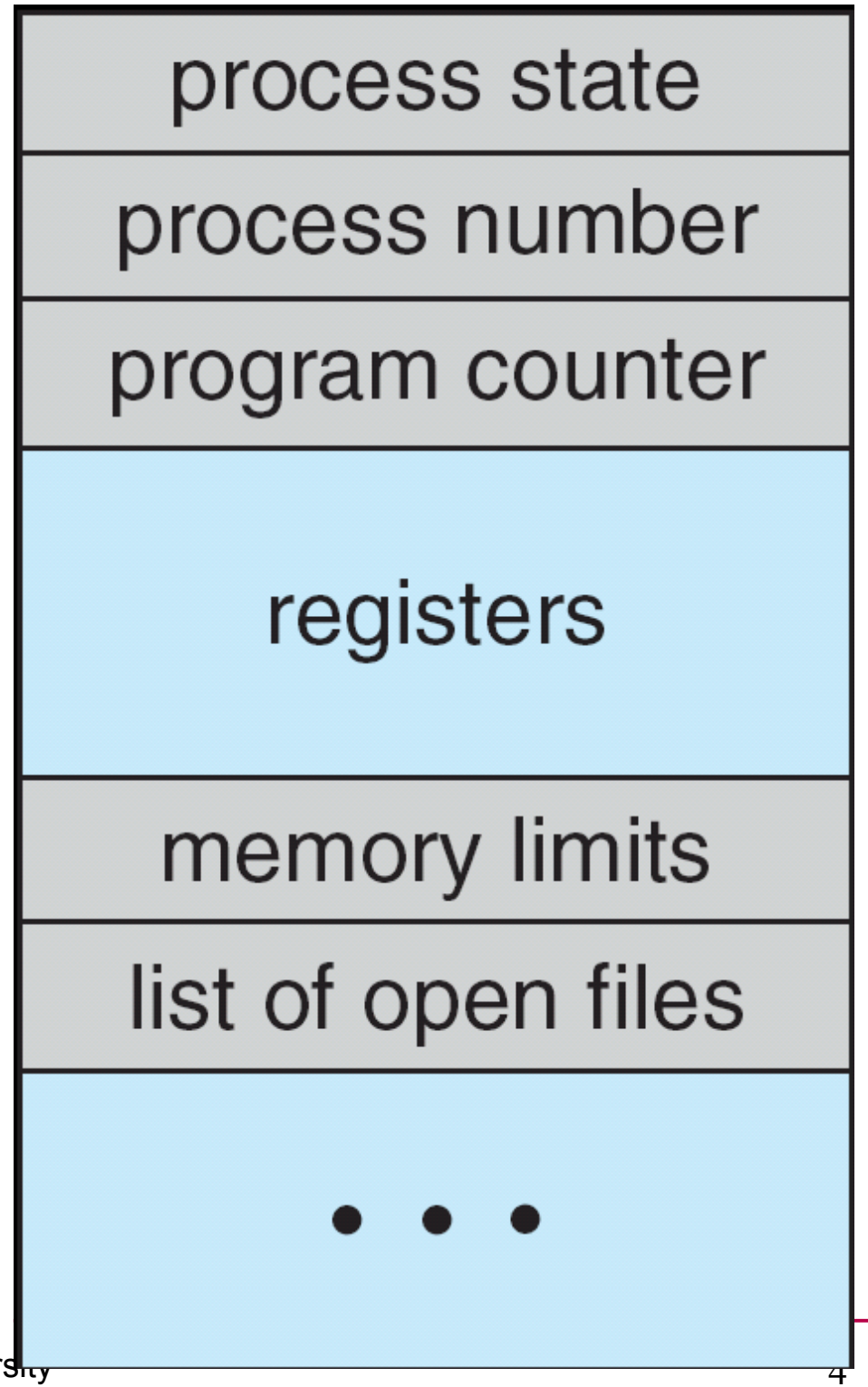
State Machine



- 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

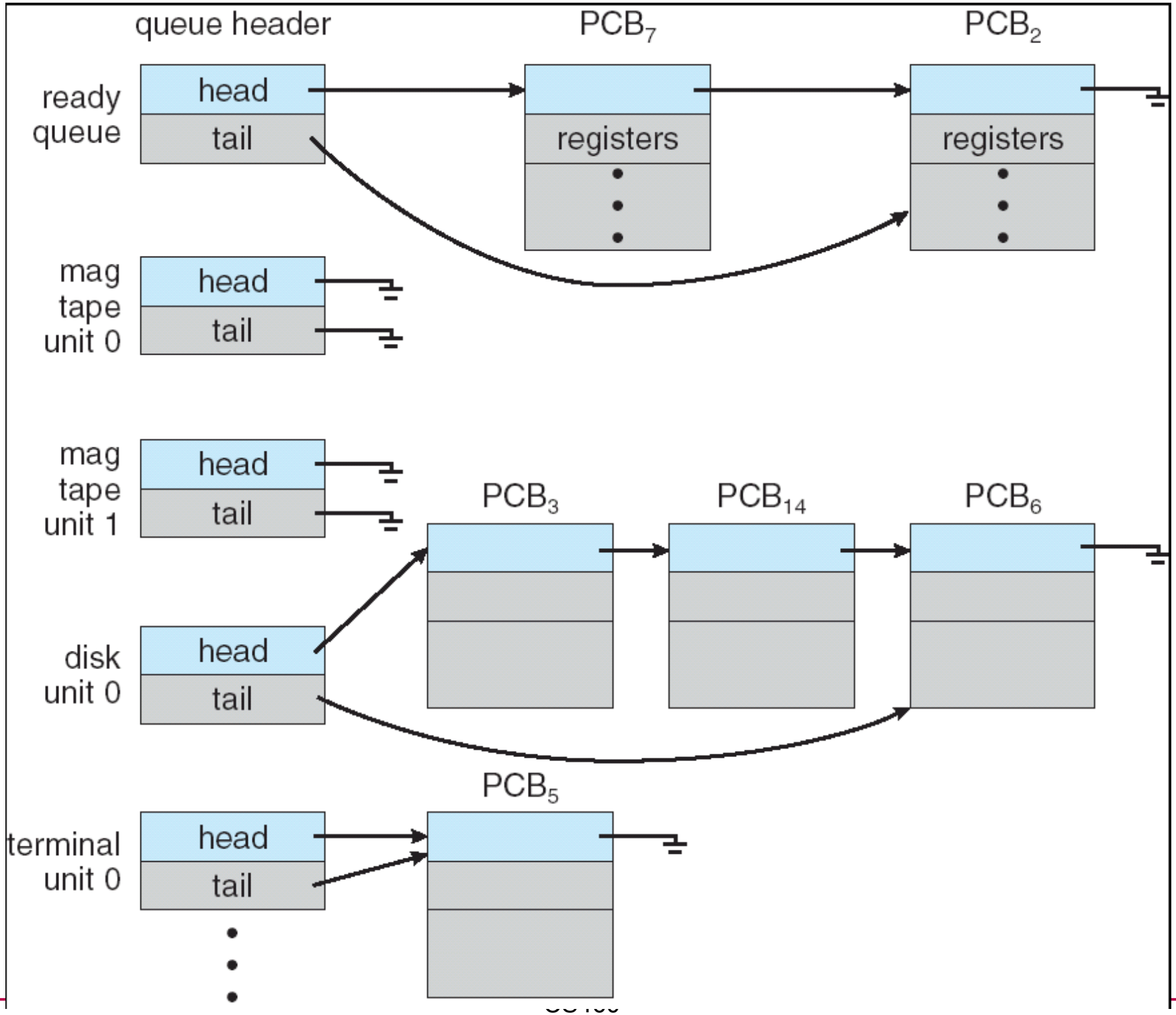
- Process Scheduler
  - Purpose:
  - Data structures:
  - Dispatched:

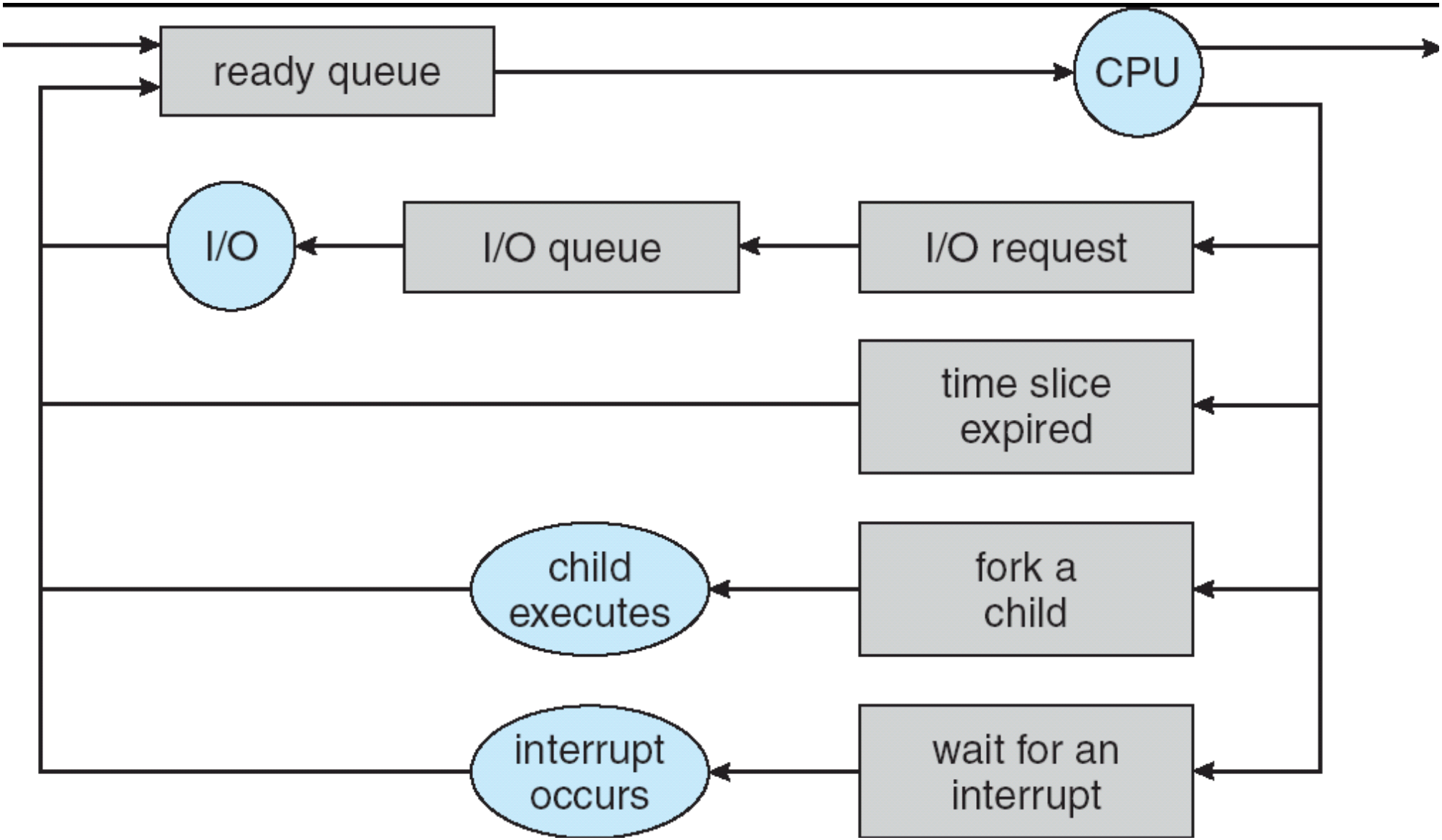
Schedulers

- 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

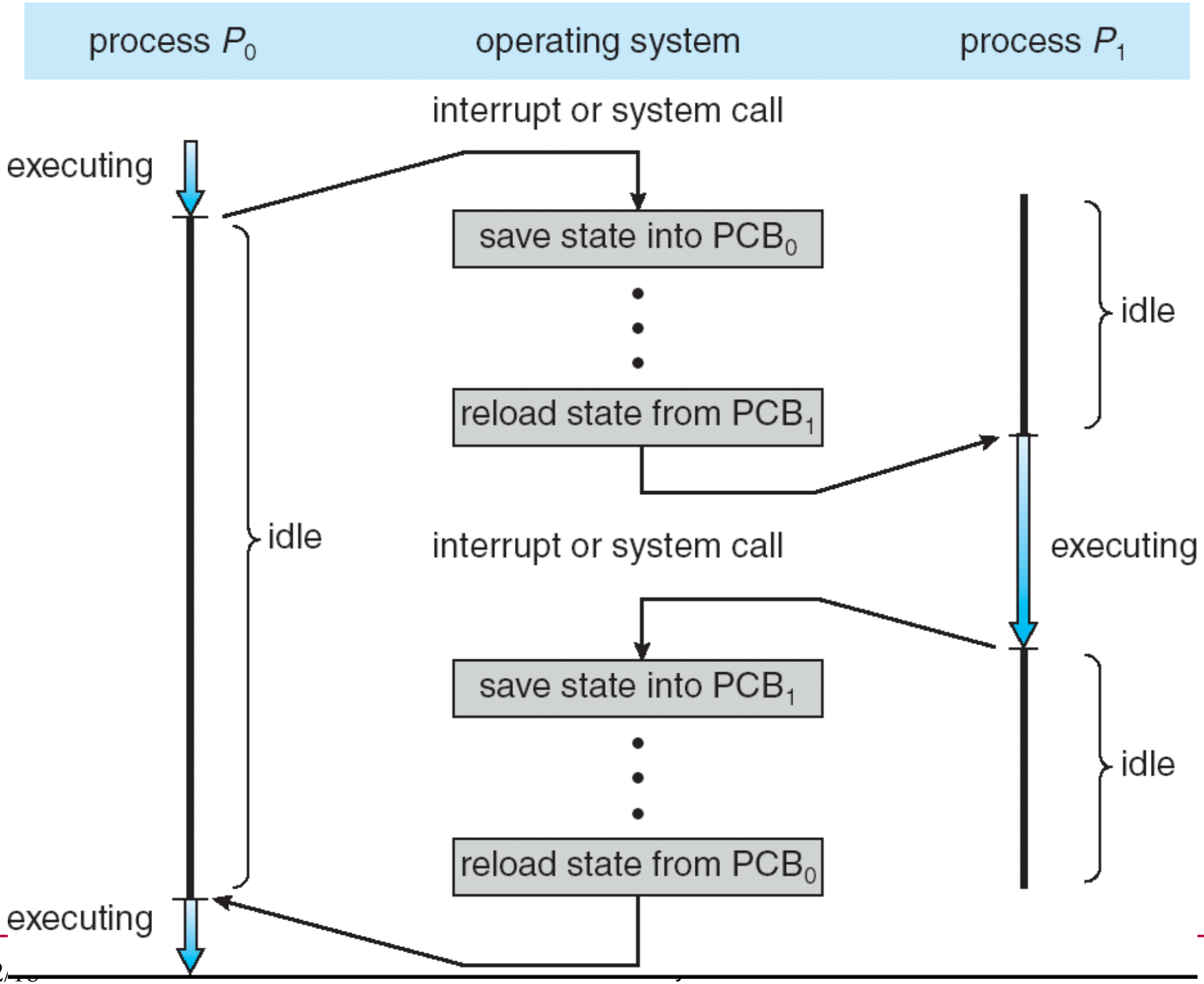

ready queue
CPU
I/O
I/O queue
I/O request
time slice expired
child executes
fork a child
interrupt occurs
wait for an interrupt

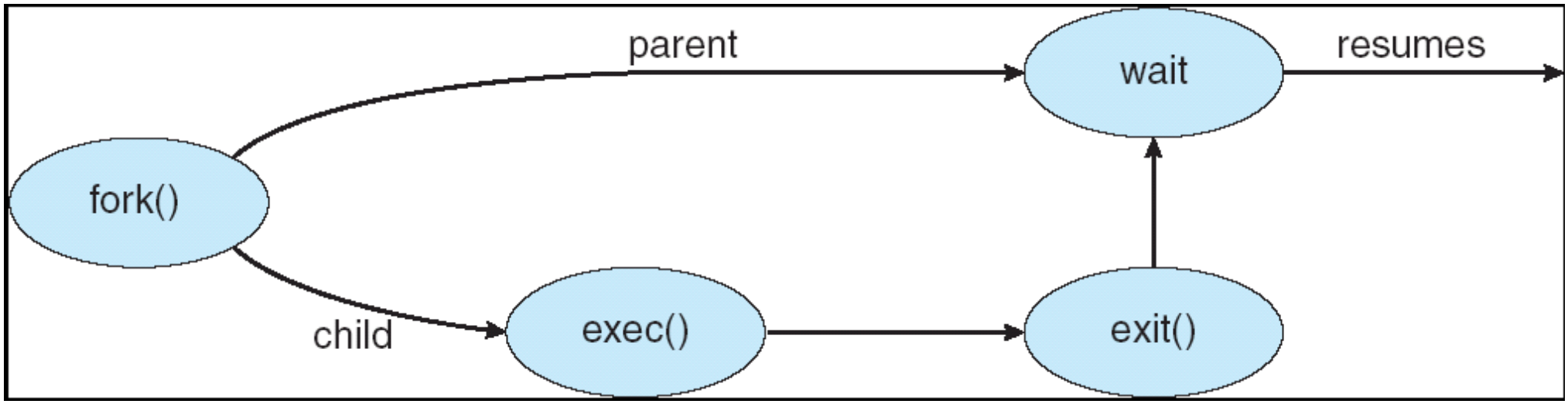
Context Switch

- 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
reload state from PCB_1
idle
idle
interrupt or system call
executing
save state into PCB_1
reload state from PCB_0
idle
executing


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

Process Creation

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
    pid_t  pid;
    int value = 0;

    value = 9;

    /* fork another process */
    pid = fork();

    fprintf(stderr,"The value: %d pid: %d\n\n", value, pid);

    if (pid < 0)
    {
        /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
    }
```

Process Creation

```
    else if (0 == pid)
    {
        /* I AM A CHILD */
        fprintf(stderr,"The value: %d pid: %d\n\n", value, pid);

        value = 10;

        fprintf(stderr,"The value: %d pid: %d\n\n", value, pid);


        execlp("/usr/bin/echo", "echo", "HELLO", NULL);
    }
    else
    {
        /* I AM A PARENT */
        wait(NULL);
        printf ("\n\nChild Complete value: %d pid: %d\n\n",
            value, pid);
    }
    return 0;
}
```

Process Termination

- kill(pid, signal)

  $ man kill

  $ ps u

  $ kill -9 pid

  $ man -s 2 kill

  $ man -s 7 signal

- Cascading termination:

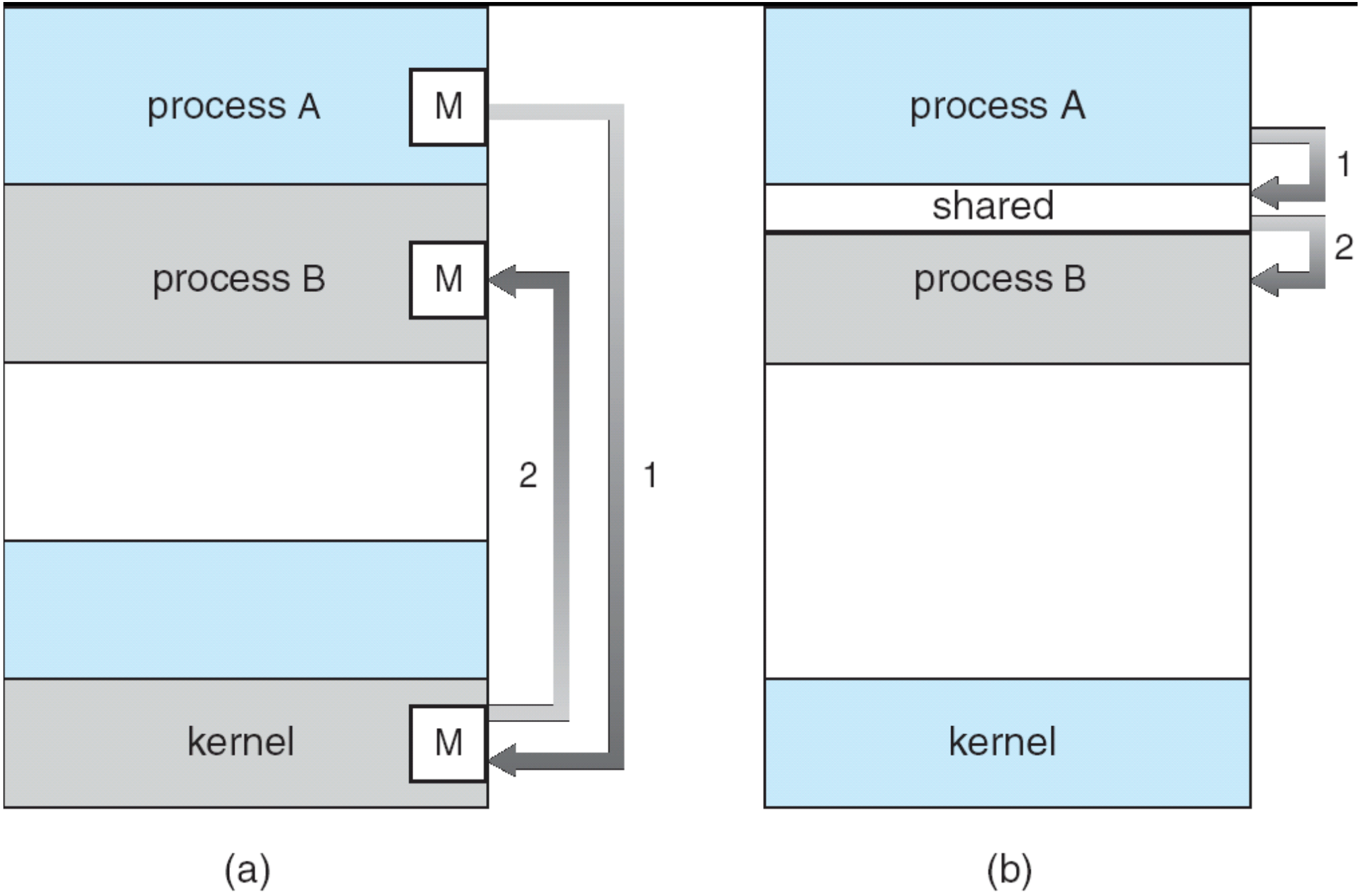
Windows (Win32 API)

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

Interprocess Communication

- Why do we want this?

- Types:
  - Shared memory:
  - Message passing:

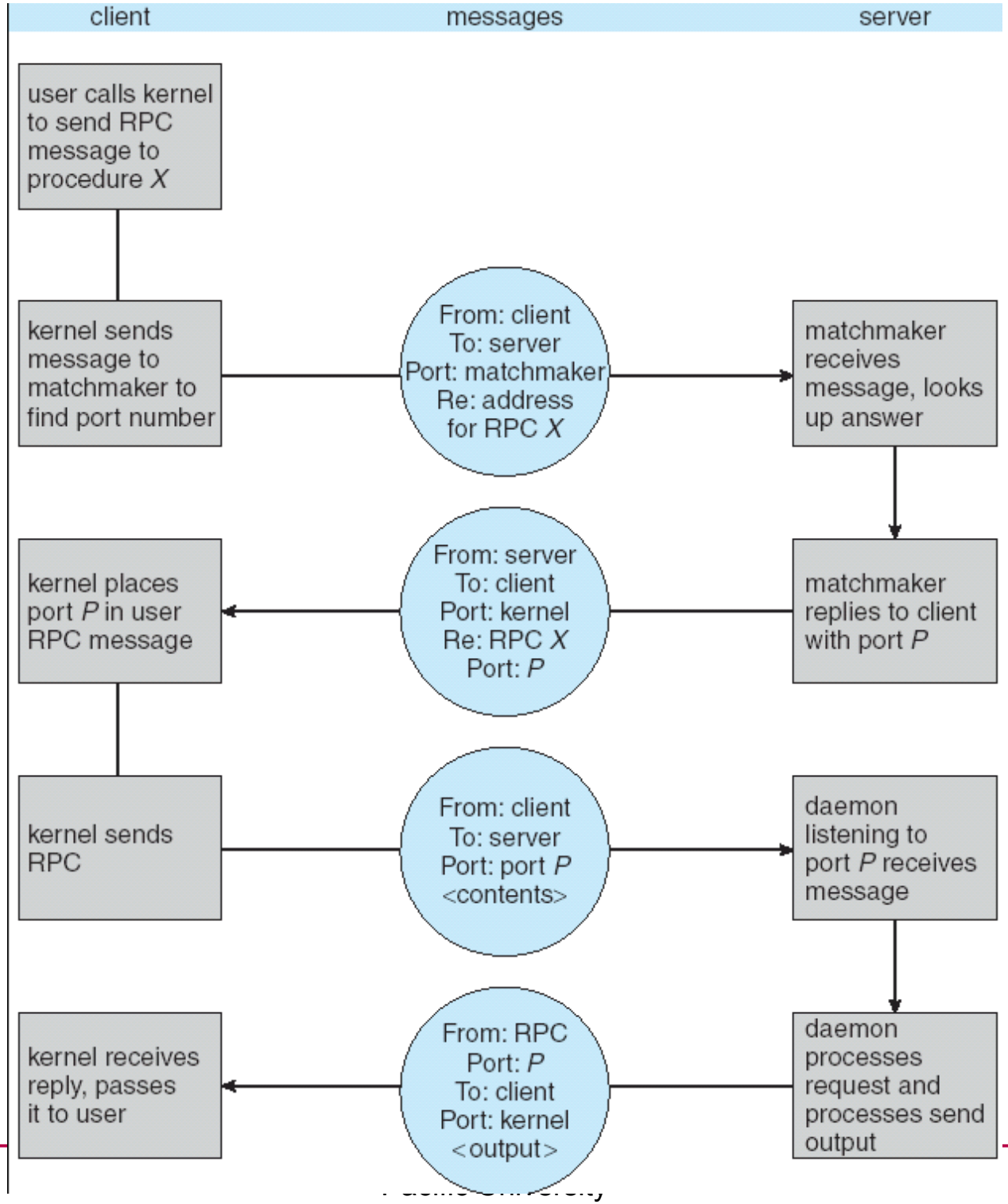

process A
M
process B
M
2
1
kernel
M
(a)
process A
1
shared
2
process B
kernel
(b)

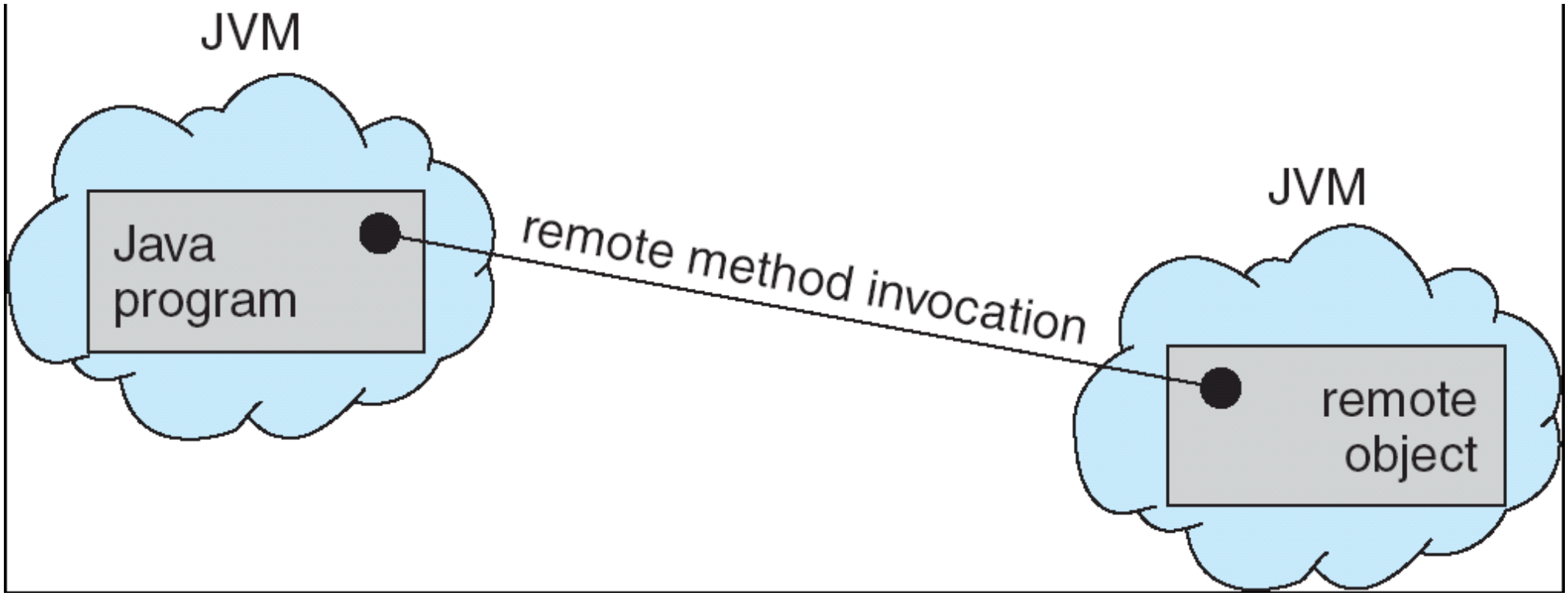
Shared Memory

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <fcntl.h>

int main()
{
    int segment_id;
    char *shared_memory;

    const long size = sysconf(_SC_PAGESIZE);

    pid_t pid;

    /* 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);

    pid = fork();
```

Shared Memory

```
if ( 0 != pid)
{
    /* Write to shared memory */
    sprintf(shared_memory, "Hello CS 460!");

    /* for child to terminate */
    wait(NULL);

    /* detach from shared memroy */
    /* the child should have already detached before termination */
    shmdt(shared_memory);

    /* remove shared memory */
    shmctl(segment_id, IPC_RMID, NULL);
}
else
{
    /* wait for parent to write */
    /* hacky synchronization */
    sleep (2);

    /* read/print from shared segment */
    printf("MESSAGE: %s\n", shared_memory);

    /* detach from shared memroy */
    shmdt(shared_memory);
}
return 0;
}
```


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()

```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>

#define MAXLEN 1024

int main()
{
    /* dup2(int oldfd, int newfd)  makes newfd be
     * the copy of oldfd closing newfd first if necessary.
     */

    char data[MAXLEN];
    int fd;
    int save_stdout_fd;

    save_stdout_fd = dup(STDOUT_FILENO);

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

    dup2(fd, STDOUT_FILENO);

    // stderr
    fprintf(stderr,"> ");

    // stdin
    fgets(&(data[0]),MAXLEN,stdin);
```

dup2()

```
    // test.txt via fd
    write(fd, &(data[0]), strlen(data));

    // ????
    printf("%d: %s\n", __LINE__, data);

    // ????
    close(fd);

    // ????
    printf("%d: ONCE MORE: %s\n", __LINE__, data);

    // ????
    dprintf(save_stdout_fd, "%d: SCREEN!\n\n", __LINE__);

    // why is this necessary?
    fflush(NULL);

    dup2(save_stdout_fd, STDOUT_FILENO);

    // ????
    printf("%d: FINALLY: %s \n", __LINE__, data);

    return 0;
}
```

dup2()

Let's investigate with gdb .....

```
(gdb) break dup2
Breakpoint 1 at 0x400720

(gdb) disas 0x400720
Dump of assembler code for function dup2@plt:
   0x0000000000400720 <+0>:     jmpq   *0x20090a(%rip)
   0x0000000000400726 <+6>:     pushq  $0x3
   0x000000000040072b <+11>:    jmpq   0x4006e0
End of assembler dump.

(gdb) run
Breakpoint 1, 0x00007ffff7b18a80 in dup2 () from /lib64/libc.so.6

(gdb) disas $pc
Dump of assembler code for function dup2:
=> 0x00007ffff7b18a80 <+0>:     mov    $0x21,%eax
   0x00007ffff7b18a85 <+5>:     syscall
   0x00007ffff7b18a87 <+7>:     cmp    $0xfffffffffffff001,%rax
   0x00007ffff7b18a8d <+13>:    jae    0x7ffff7b18a90 <dup2+16>
   0x00007ffff7b18a8f <+15>:    retq
   0x00007ffff7b18a90 <+16>:    mov    0x2bd3d1(%rip),%rcx
   0x00007ffff7b18a97 <+23>:    neg    %eax
   0x00007ffff7b18a99 <+25>:    mov    %eax,%fs:(%rcx)
   0x00007ffff7b18a9c <+28>:    or     $0xffffffffffffffff,%rax
   0x00007ffff7b18aa0 <+32>:    retq
End of assembler dump.
```

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.

```
Dump of assembler code for function dup2:
=> 0x00007ffff7b08c70 <+0>:     mov    $0x21,%eax
   0x00007ffff7b08c75 <+5>:     syscall
(gdb) x 0x00007ffff7b08c75
0x7ffff7b08c75 <dup2+5>:   0x3d48050f
```

0x21 = ?

arch/x86/include/generated/uapi/asm/unistd_64.h

```
#define __NR_dup 32
#define __NR_dup2 33
```

arch/x86/entry/syscalls/syscall_64.tbl

```
32  common  dup         sys_dup
33  common  dup2            sys_dup2
```

https://en.wikibooks.org/wiki/X86_Assembly/Interfacing_with_Linux#Making_a_syscall

simple pipe()

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

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)
{
    memset(data, '\0', maxSize);
    fprintf(stderr,"> ");
    fgets(data, maxSize, stdin);
}

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(0 == childPid)
{
    /* 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]);
}
```

ELF: Executable and Linkable Format

- Executable file
  - a non-running process on disk!
- Tools
  - readelf
  - objdump

Resources

- http://www.cs.stevens.edu/~jschauma/810/elf.html
- refspecs.linuxfoundation.org/elf/x86_64-abi-0.99.pdf
- refspecs.linuxfoundation.org/elf/gabi4+/contents.html
- skyfree.org/linux/reference/ELF_Format.pdf
- www.muppetlabs.com/~breadbox/software/ELF.txt
- www.muppetlabs.com/~breadbox/software/tiny/teensy.html
- github.com/BR903/ELFkickers

- iecc.com/linker/linker10.html
  - iecc.com/linker (number: 1, 3, 10)
- users.eecs.northwestern.edu/~kch479/docs/notes/linking.html