Lecture 2
Processes

I233E OPERATING SYSTEMS

RAZVAN BEURAN

Today’s Topics

Processes

- Why they exist
- What they are
- How they work

Operations on processes

- Creation
- Termination

Inter-process communication

- Why it is needed
- How it works

Processes

Users

- Multiple activities

System

- Must run multiple programs concurrently

Example

- Browser + text editor + email + calculator + …

Process

- Program in execution
- Unit of work
- Similar concepts: job, task, user program

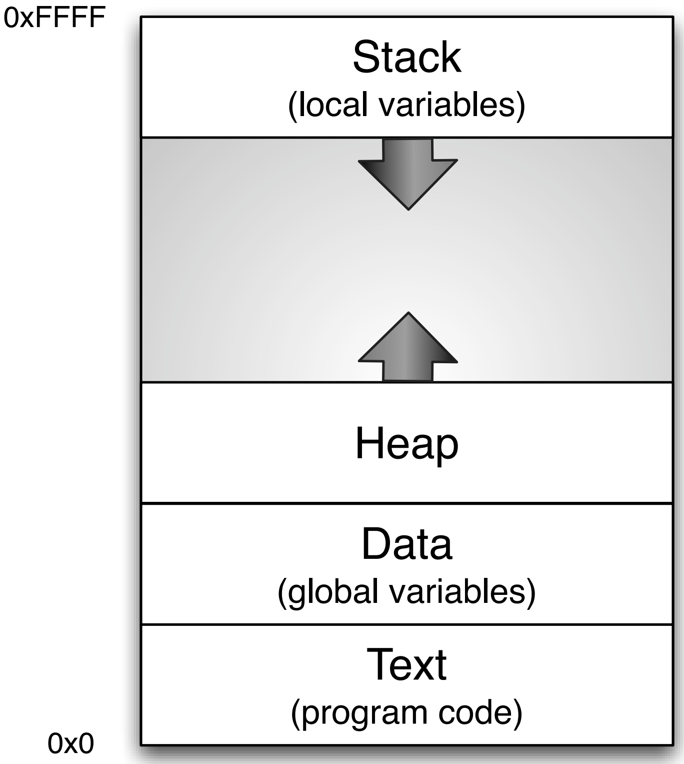
Process

What is a process?

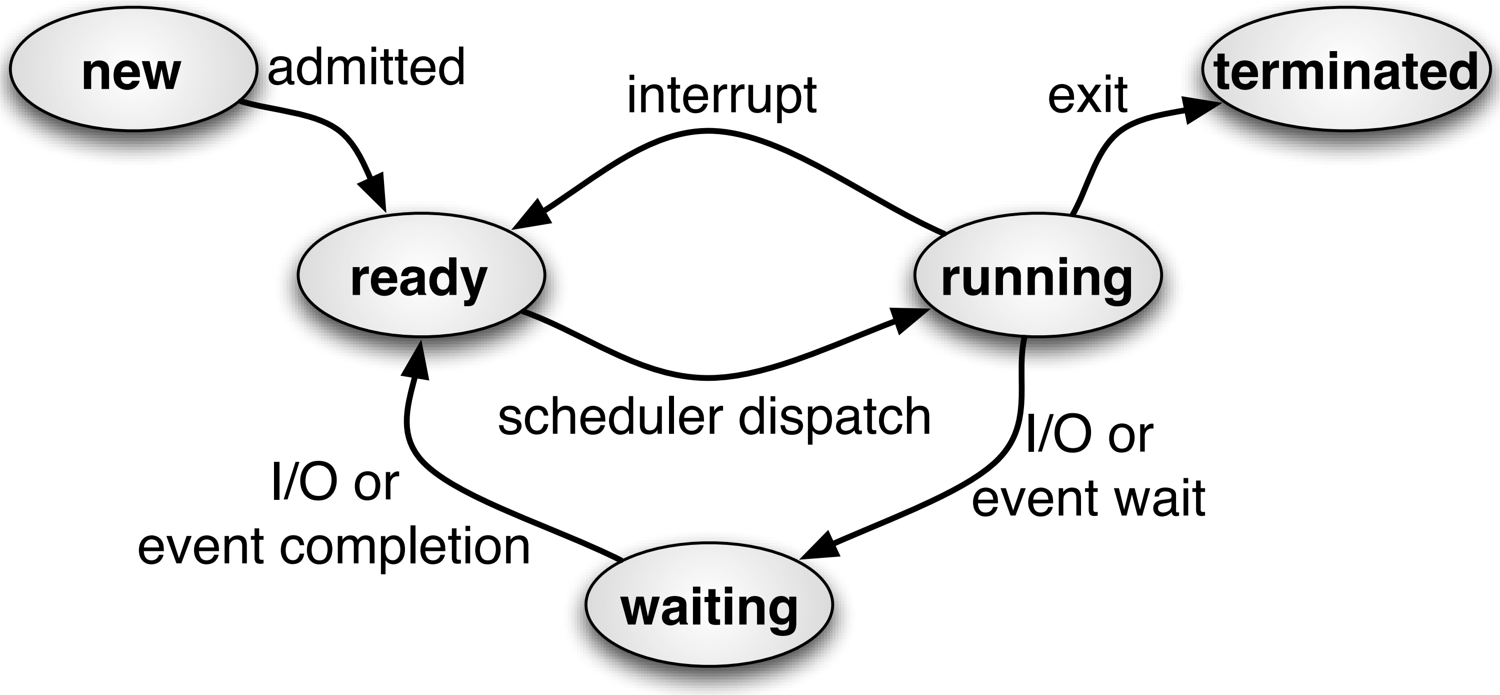
- Program code (text section)
- Data section (global variables)
- Process heap
- Process stack
- Current activity

Current activity

- Program Counter (PC)
- Content of process registers



Process States



Process Control Block (PCB)

Process state

Process number (id)

Program counter & registers

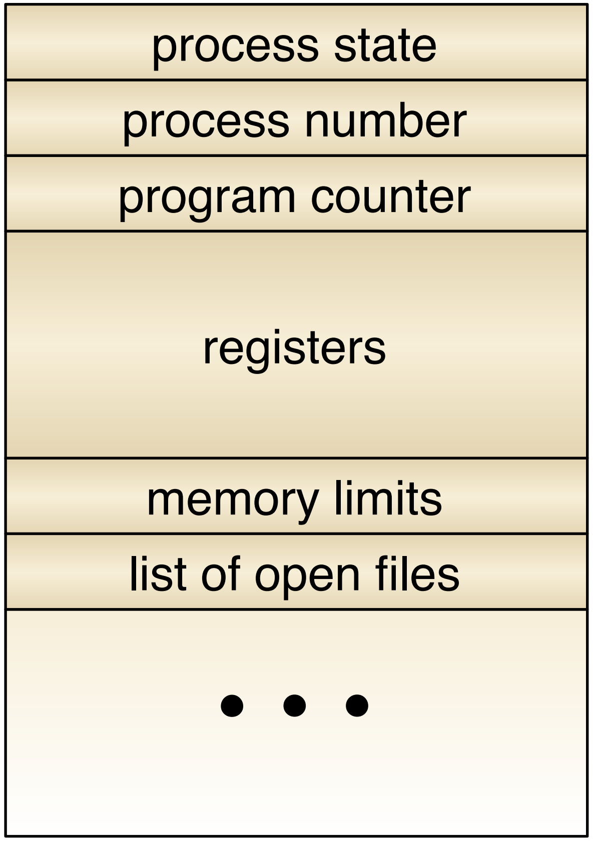
Memory management info

I/O status info

Accounting info

Scheduling info

Etc.



process state

process number

program counter

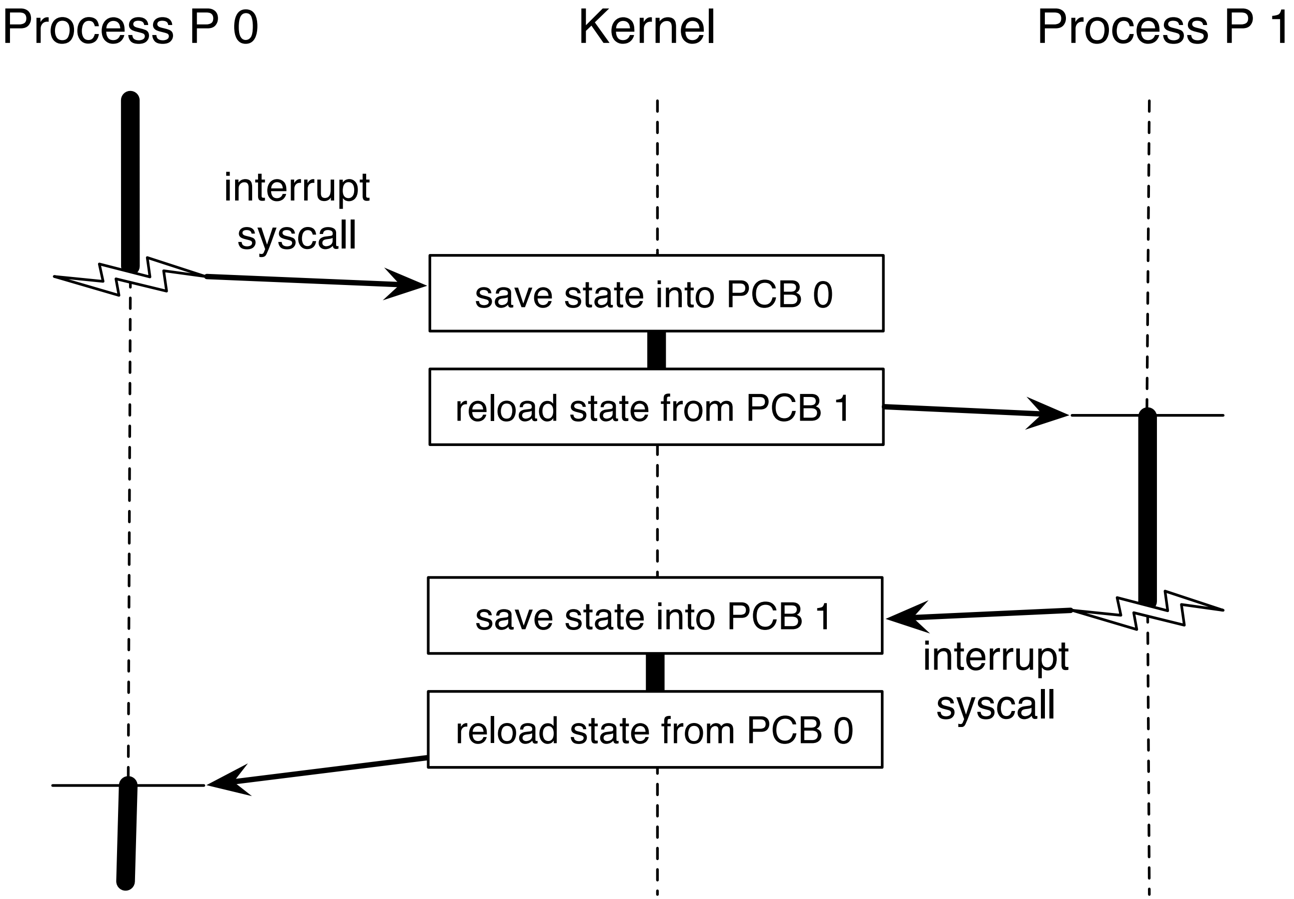
registers

memory limits

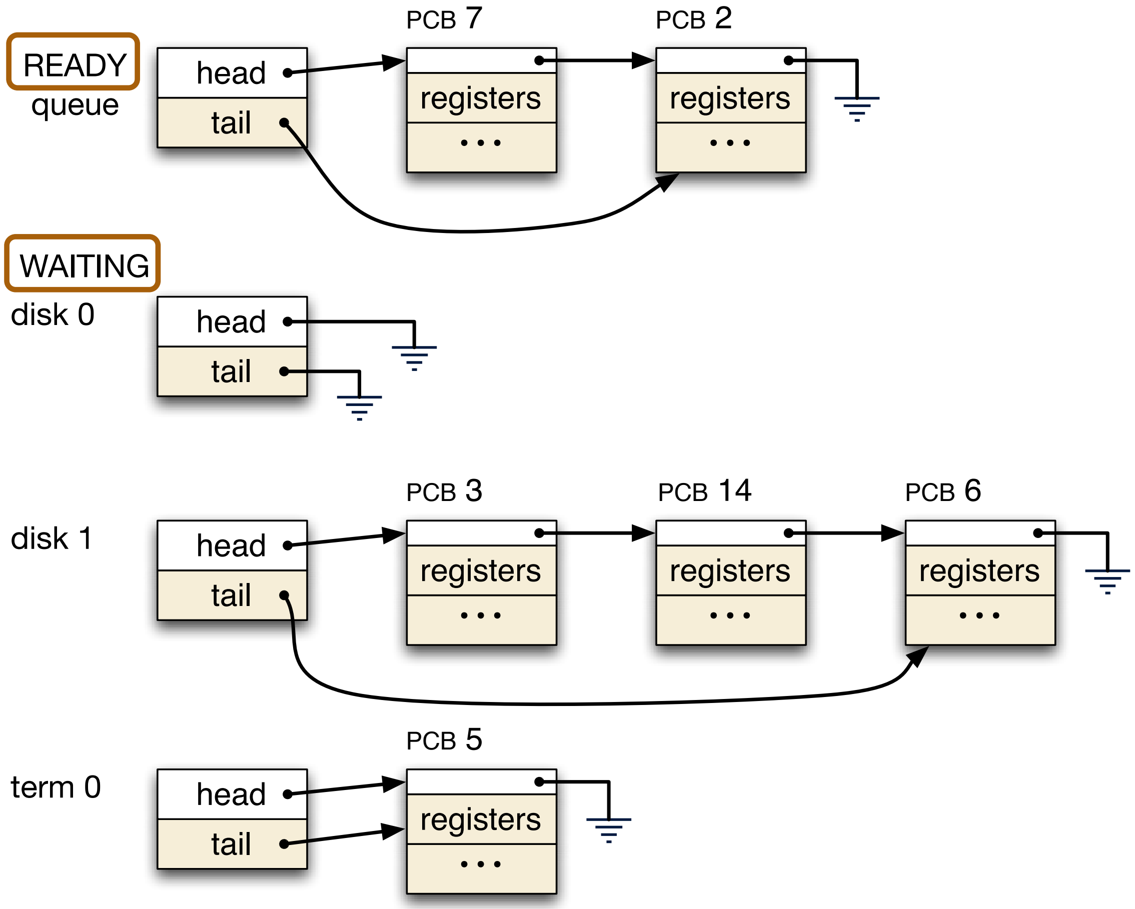
list of open files

• • •

Process Switching

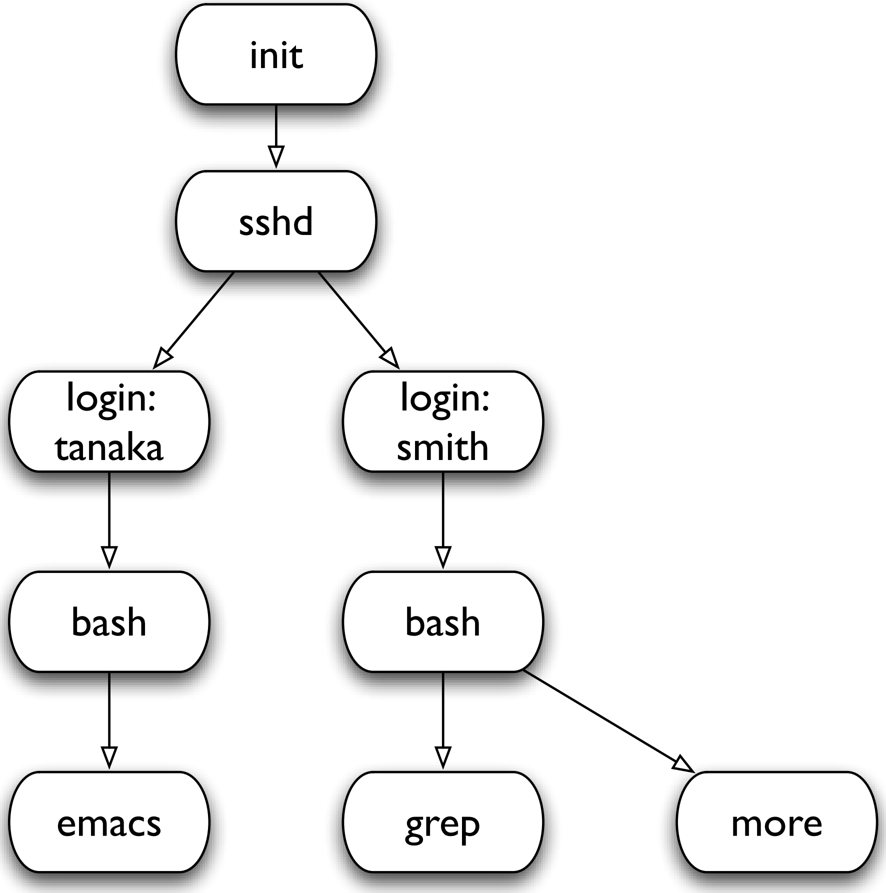


Process Scheduling



Operations on Processes

Process Tree



Process Creation

Create process

- `pid = fork ()`

Check return status

- `pid < 0` => error
- `pid == 0` => child process actions
- `pid > 0` => parent process actions (`pid` is ID of child process)

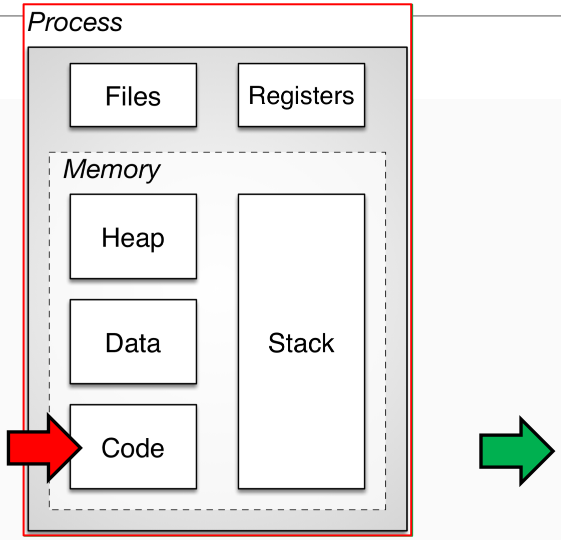
Parent waits for child to terminate

- `wait(NULL)`

Process Creation Example

```c
// #include directives omitted

int main(int argc, const char* argv[]) {
  int pid;
  /* Fork another process */
  pid = fork();
  if(pid < 0) {
    /* Error occurred */
    fprintf(stderr, "Fork failed!\n");
    return -1;
  } else if (pid == 0) {
    /* Inside the child process */
    printf ("I am the child process (pid:%d).\n", getpid());
    execlp("/bin/ls", "ls", "-l", NULL); /* Run the command 'ls' */
    return 0;
  } else {
    /* Inside the parent process */
    wait(NULL); /* Wait for the child process to end */
    printf("I am the parent process (pid:%d); my child (pid:%d) has completed.\n", getpid(), pid);
    return 0;
  }
}
```



DEMO

Process Termination

Process terminates when

- Returns from main()
- Calls function exit()
- Error occurs

Return status

- Exit value

Inter-Process Communication

Inter-Process Communication (IPC)

For process cooperation

- Information sharing
- Computation speed-up
- Modularity
- Convenience

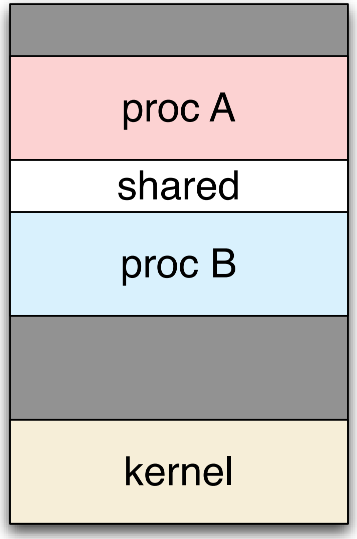
Models

- Shared memory
- Message passing

Shared Memory

POSIX Shared Memory

- Create a memory segment
  ```
  segment_id = shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);
  ```
- Obtain a memory segment
  ```
  shared_memory = (char *) shmat(segment_id, NULL, 0);
  ```
- Use the shared memory
  ```
  sprintf(shared_memory, “Hello World!\n”);
  ```
- Release memory segment
  ```
  shmdt(shared_memory);
  ```



Message Passing

Pipes

- Prepare pipe
  `pipe(fd);`
- Write data to pipe
  `write(fd[1], MESSAGE, MESSAGE_LENGTH)`
- Read data from pipe
  `read(fd[0], &buffer, BUFFER_LENGTH)`

Close unused ends of the pipes to prevent wastage

- In reader process
  `close(fd[1]);`
- In writer process
  `close(fd[0]);`

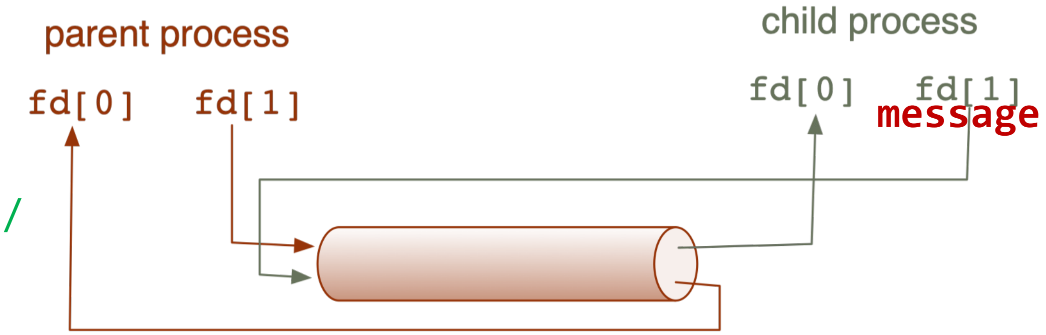
Message Passing Example

```
int pid, fd[2];

pipe(fd); /* Prepare pipe */

pid = fork();

if (pid == 0) { /* Child process */
    close(fd[0]);
    write(fd[1], “message”, 8);
    // ...
} else { /* Parent process */
    close(fd[1]);
    read(fd[0], &buffer, BUFFER_SIZE);
    // ...
}
```



Writer

Reader

Summary

Processes

- Needed to allow parallel execution of tasks
- OS switches between processes based on scheduling algorithms (scheduling will be discussed later in the course)

Operations on processes

- Creation
- Termination

Inter-process communication

- Shared memory
- Message passing

Next Time

Threads

Threading models

Thread libraries