PROCESS MANAGEMENT

PREVIOUS VIDEO

- Introduction to operating system
- Kernel vs operating system
- Functions of operating system
- Examples of operating system and kernel
- System calls
- Interrupts
- Video Link + telegram link + form link in the description

AGENDA

- Program VS process
- Process control block
- Process states
- Process creation through fork()
- Exec(), wait() and exit() system calls
- Zombie and orphan process

INTERVIEW QUESTIONS

- Difference between a program and a process
- Which data structure is used to represent a process?
- List some important fields in Process Control Block
- What is INIT process in Linux?
- What are the different states of a process? Explain with process state diagram.
- How will you create a new process in Linux?
- How does a parent process differ from a child process?
- Number of processes created with N fork() calls?

INTERVIEW QUESTIONS CONTINUED

- What is the use of exec(), wait() and exit() system calls?
- What happens if parent doesn’t wait for its child process?
- Give some real world examples where fork(), exec() and wait() system calls are used?
- What is zombie process? What are the side effects of zombie processes?
- What is an orphan process? How it differs from zombie process?
- Give examples of zombie and orphan process.

PROGRAM VS PROCESS

- Program – An executable file that is stored on the hard disk or a secondary memory
- When you execute this file either by double clicking or from command line, it becomes a process
- Program – Passive entity
- Process – Active entity
- Process memory = stack + data + heap + text
- Program is just the text section of memory
- Process is text section + a lot of other attributes

What are other attributes of a process?

- Program counter(PC) – Address of next instruction to be executed on CPU
- Process states – new, ready, running, waiting etc.
- CPU registers – value of CPU registers are stored when context switch happens
- List of open files that the process is using for reading and writing
- Process priority – Used in process scheduling
- Process ID – unique identifier for each process
- Parent process ID – Process ID of parent process
- Memory information – stack, heap, data, text

PROCESS CONTROL BLOCK

- These all attributes or information about a process are stored in a data structure called Process Control Block(PCB).
- Struct task_struct {

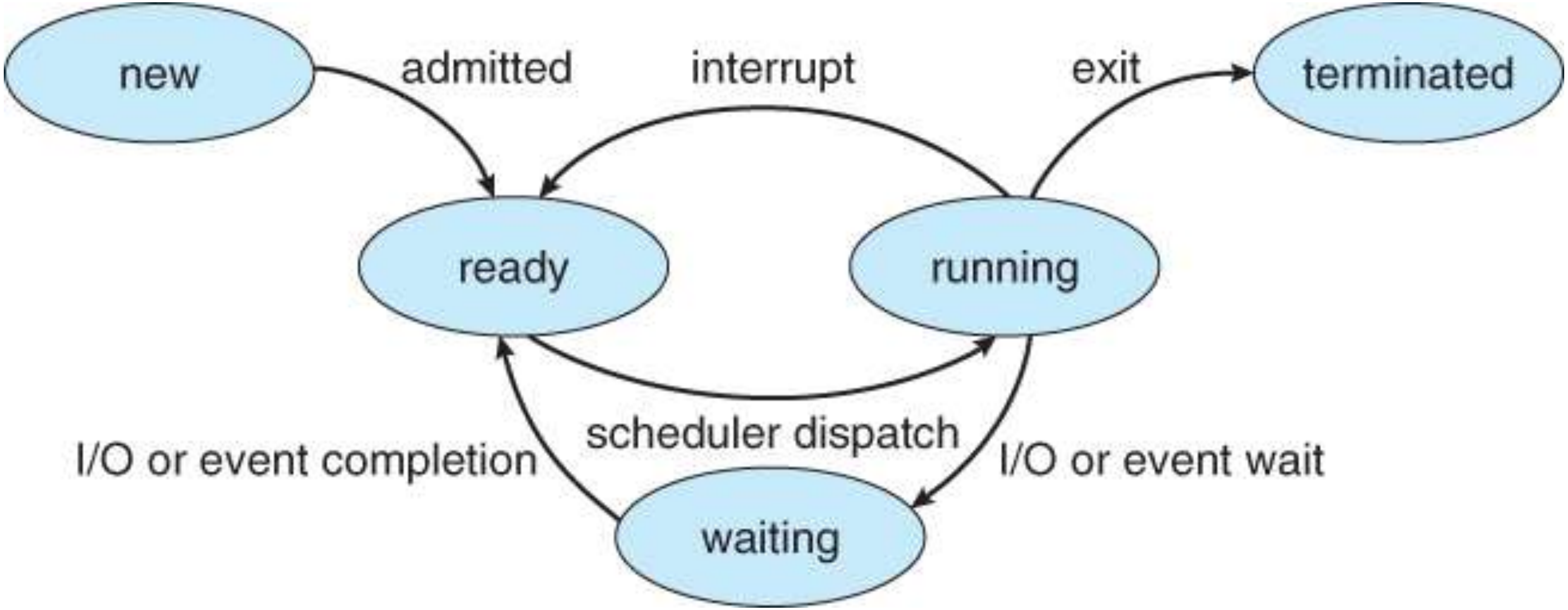
```
             pid_t pid;
            long state;
             int priority;
             struct task_struct *parent;
             struct files_struct *files;
             struct mm_struct *mm;
             long program_counter;
            .
             .
     }
```

- Video: https://youtu.be/x2JIEgAVkTI

PROCESS STATES

- Batch operating system in old times
- Jobs were submitted in batches
- Jobs were first stored in job queue on disk
- Not all job can fit in main memory
- Selected jobs were loaded into main memory by long term scheduler
- These were kept in ready queue and scheduled by CPU scheduler or short term scheduler
- Video: https://youtu.be/mhKr6Dnf2fA

PROCESS STATE DIAGRAM



WHAT EACH STATE MEANS?

- NEW
    - BATCH OS – Job arrived in the job queue
    - Time sharing OS – process is being created
- READY – Process waiting in ready queue for being assigned to a CPU/processor
- RUNNING – Program is being executed on the CPU
- WAITING – Process is waiting for some event to occur (I/O)
- TERMINATED – Process has finished its execution.

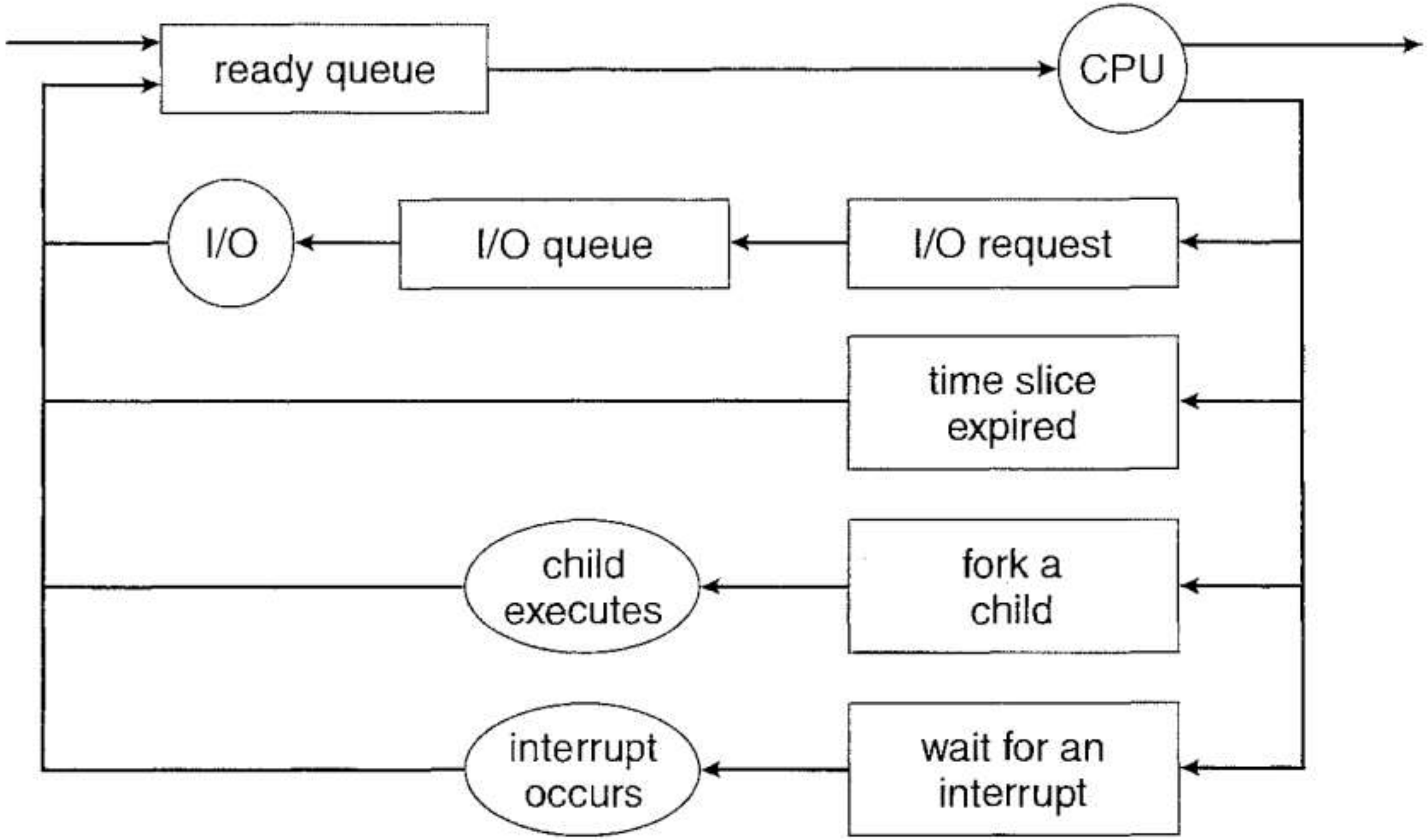
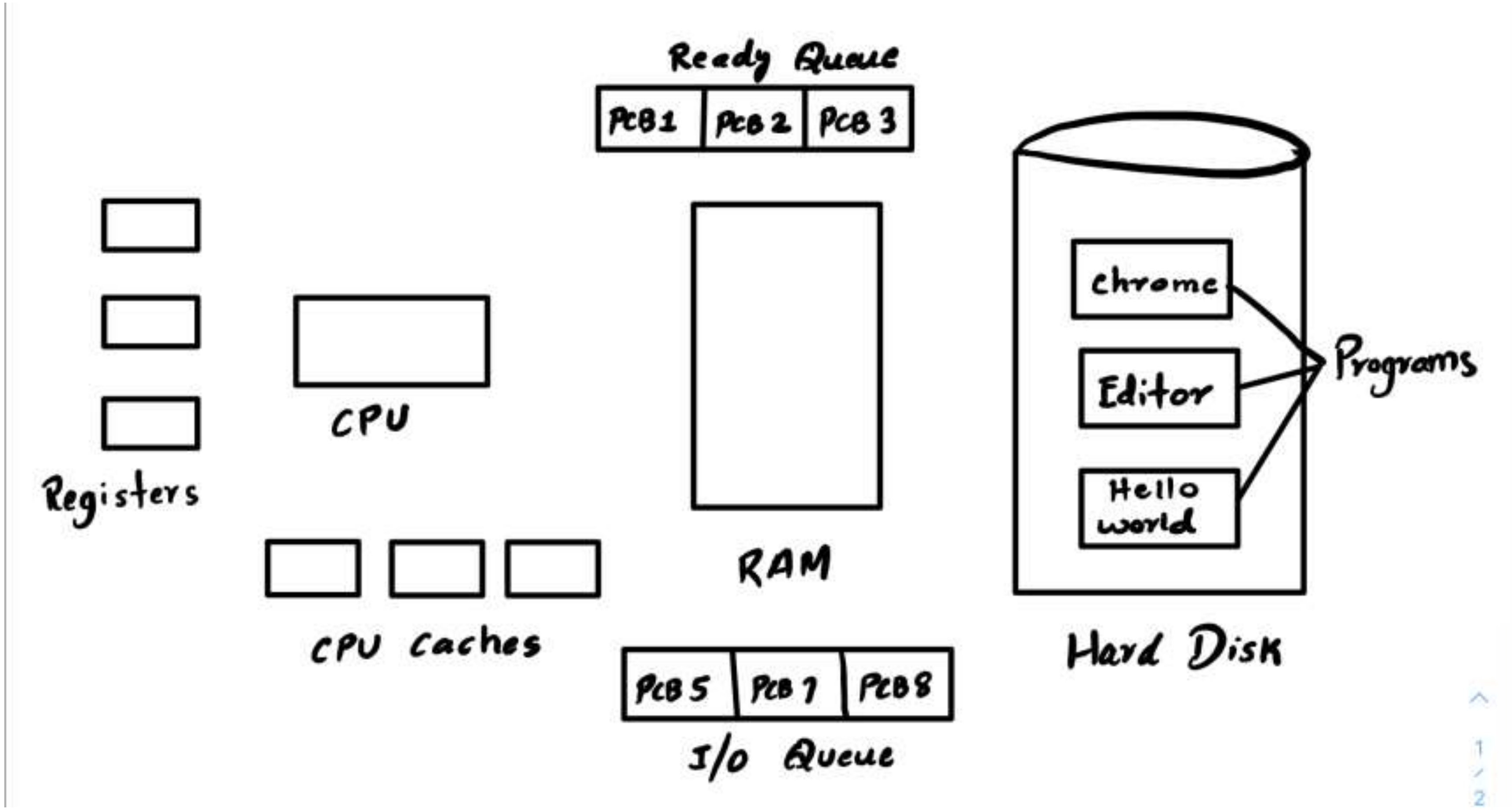


Figure 3.7 Queueing-diagram representation of process scheduling.

WHEN DOES TRANSITION FROM RUNNING TO READY HAPPENS?

- A hardware interrupt happens on the CPU
- A higher priority process arrives in the ready queue (preemptive scheduling)
- Time slice of the process expires (Round robin scheduling)



- Program – helloworld.c
- Compile – gcc helloworld.c
- Executable – a.out
- Execute the program
- a.out is bought to RAM from disk
- A PCB is created (PC -> address of first instruction)
- Enters ready queue
- CPU scheduler schedules it on CPU
- Running state
- CPU starts executing
- I/O – enters waiting state
- Time slice expires – ready state
- Terminated

INIT PROCESS

- First process that is started during booting of the system
- PID – 1
- This process keeps on running forever until you shut down.
- All other processes are either direct or indirect children of INIT process.
- Pstree –p command to view the process tree on linux

PROCESS CREATION USING FORK

- CreateProcess() in Windows
- Fork() creates a new child process that is a copy of the parent process
- In what ways the child process differs from parent?
  - PID
  - Parent PID
  - File locks
  - Process utilization info
- In what ways they are similar?
  - Program counter
  - Memory (unless fork() is followed by exec())
  - CPU registers
  - priority

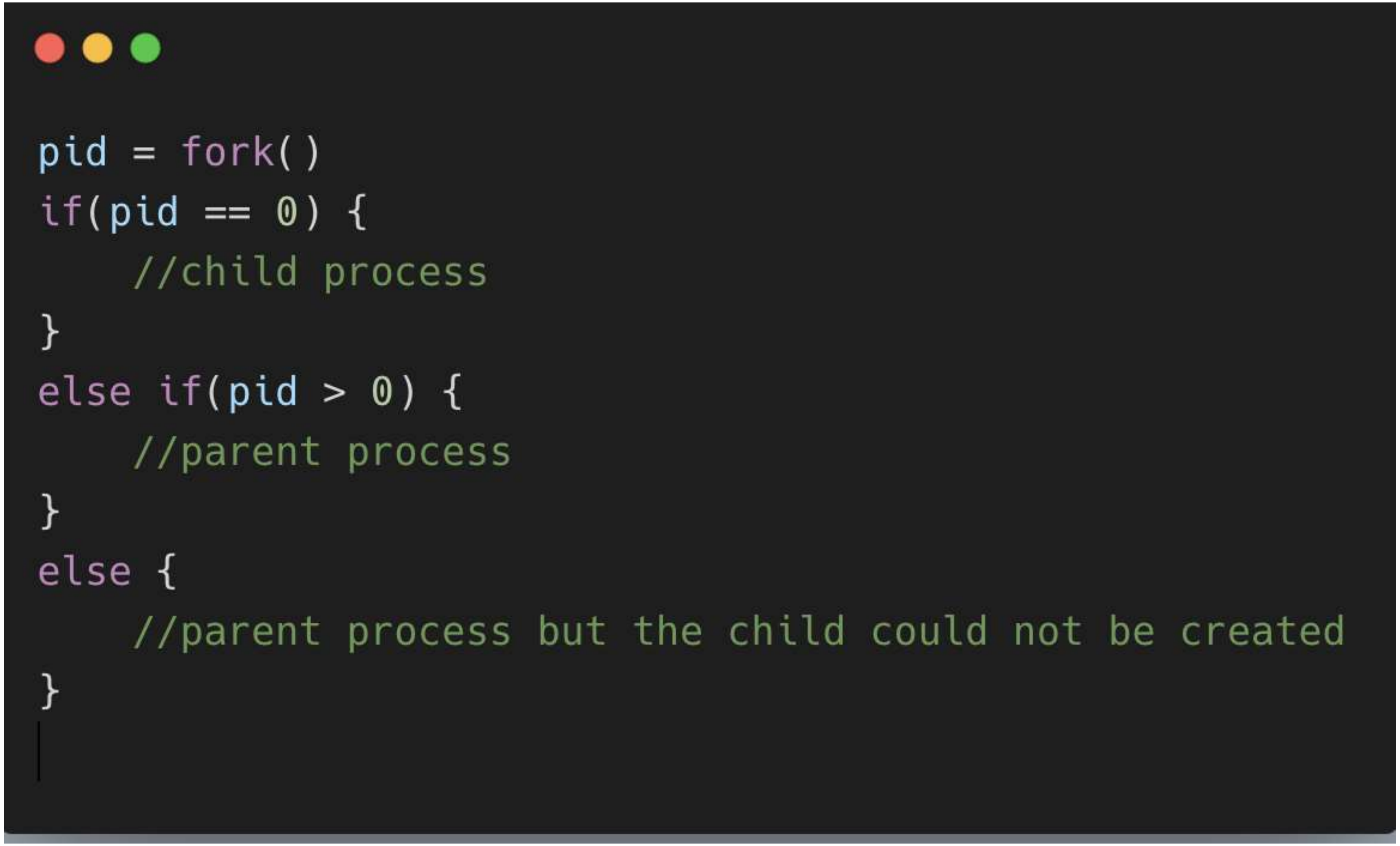
WHAT HAPPENS AFTER FORK() IS CALLED?

- Execution – Both process starts executing from the next line after fork()
- Memory – stack, heap, text and data segment are duplicated for the child process.
- Copy on write is used in modern implementation of fork.
- Copy the memory only when a write is being made to a memory page.
- In most cases, fork() is followed by exec().
- Copy on write saves expensive recopying of memory

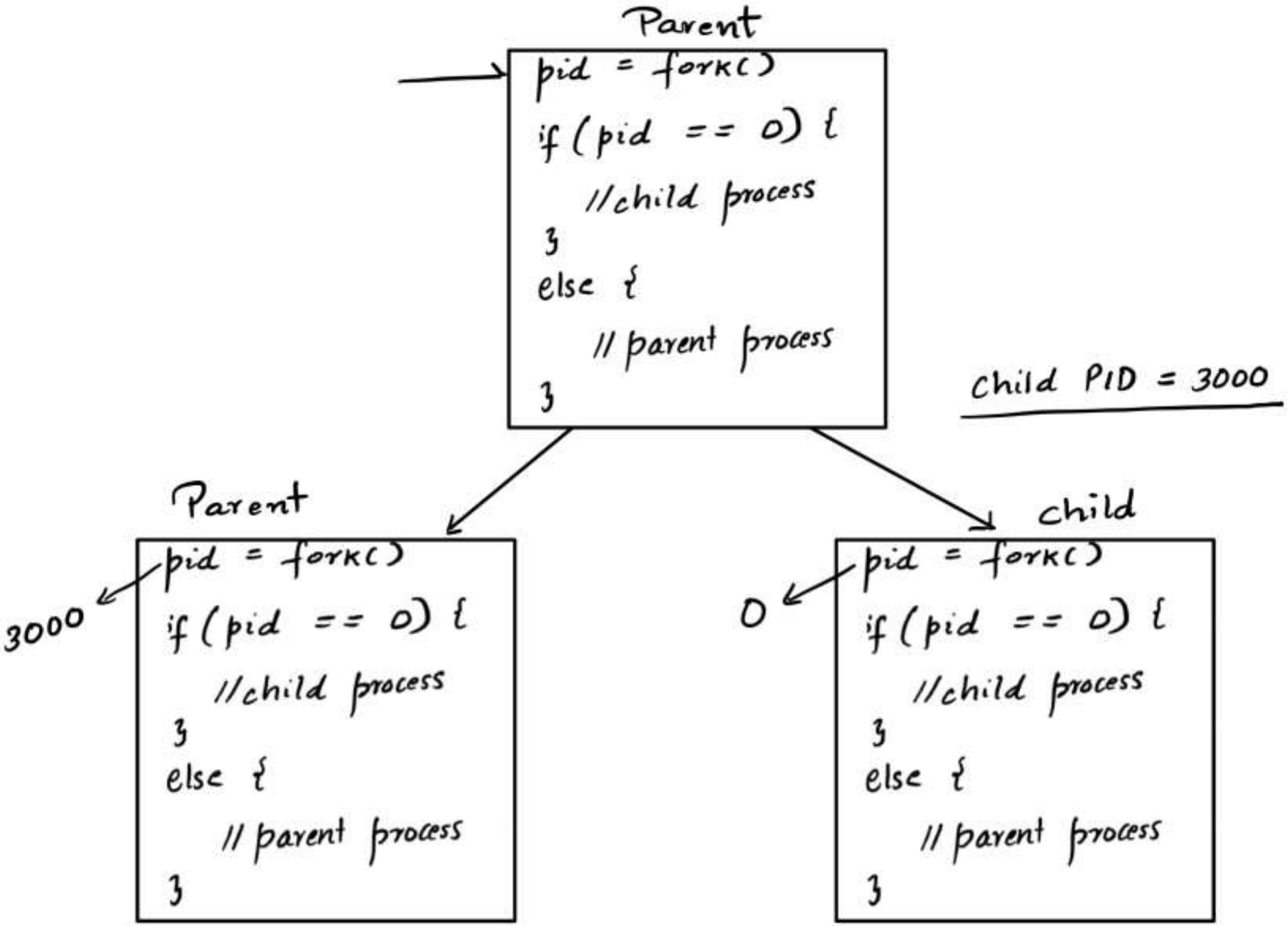
FORK CALL RETURN VALUES

- How do you differentiate between the the parent process and the child process?
- Fork() returns PID of the child process to parent
- Fork() returns 0 to the child process
- Fork() returns -1 if there is any error
- Videos
  - https://youtu.be/FXAvkNY1dGQ
  - https://youtu.be/AyZeHBPKdMs

FORK CODE EXAMPLE



```
pid = fork()
if(pid == 0) {
    //child process
}
else if(pid > 0) {
    //parent process
}
else {
    //parent process but the child could not be created
}
```



```
pid = fork()
if (pid == 0) {
    //child process
}
else {
    // parent process
}
```

```
pid = fork()
if (pid == 0) {
    //child process
}
else {
    // parent process
}
```

```
pid = fork()
if (pid == 0) {
    //child process
}
else {
    // parent process
}
```

How may child processes?

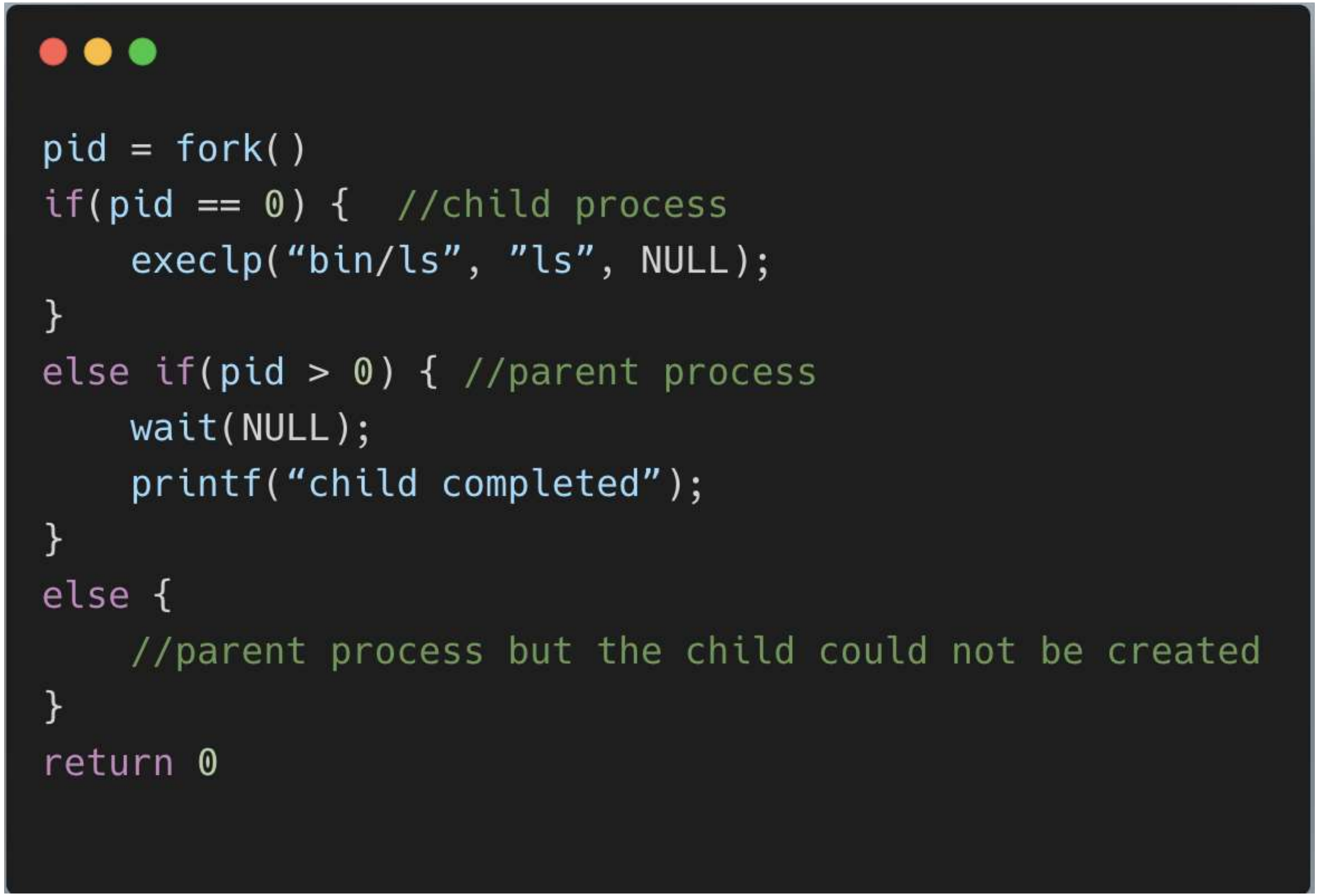
```
fork();
fork();
```

Video: https://youtu.be/iZa2vm7A6mw

What if you want the child to run another program?

- Shell program which helps in running different commands on command line
- Thousands of command will be there. Is it a good idea to put all these commands in the shell program?
- What if you want to run a program from shell that is not present in shell program? (Running your C program)
- The shell program typically uses fork() to create a new child process and asks the child process to run the command.

EXEC(), WAIT() AND EXIT() SYSTEM CALLS



```
pid = fork()
if(pid == 0) {  //child process
    execlp(“bin/ls”, ”ls”, NULL);
}
else if(pid > 0) { //parent process
    wait(NULL);
    printf(“child completed”);
}
else {
    //parent process but the child could not be created
}
return 0
```

What each system call does?

- exec() – loads a new program into the main memory and executes it.
- A process exits or terminates using exit() system call. The resources like memory/cpu of the process are set free.
- return status statement from main function = exit(status)
- PCB of the process is still present in memory
- wait() – the parent waits for the completion of the child process. Parent enters the wait state.
- The parent may be interested in knowing about the status of the child.
- After getting the status, the parent asks the kernel to clean up the PCB of child process.

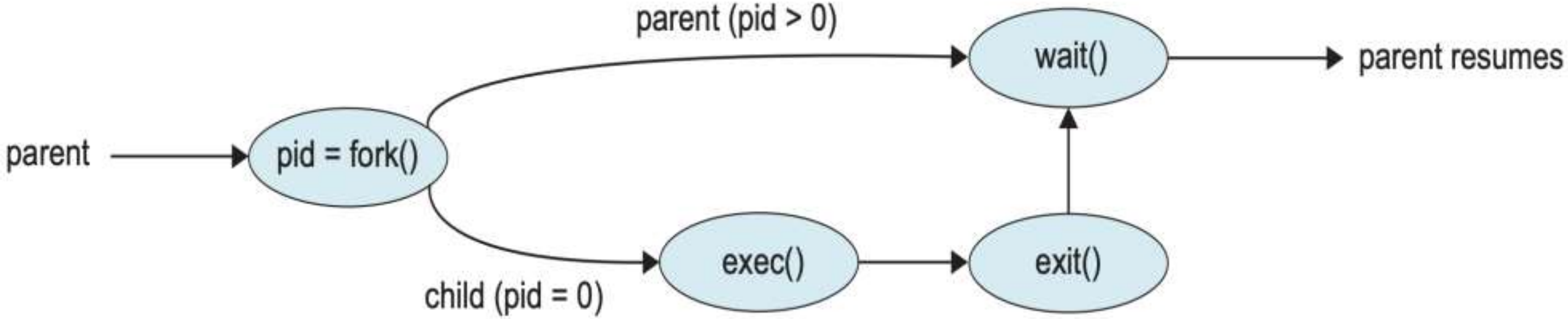


Figure 3.10 Process creation using the `fork()` system call.

What happens if the parent doesn’t wait for the child process?

- The process control block of the child remains in the process table evens after it has terminated using exit().
- These processes are called zombie process - process has terminated but PCB is still there in process table.
- Once the parent terminates, these zombie process also becomes an orphan process.
- No parent + PCB still in the process table after terminating

SIDE EFFECTS OF ZOMBIE PROCESS

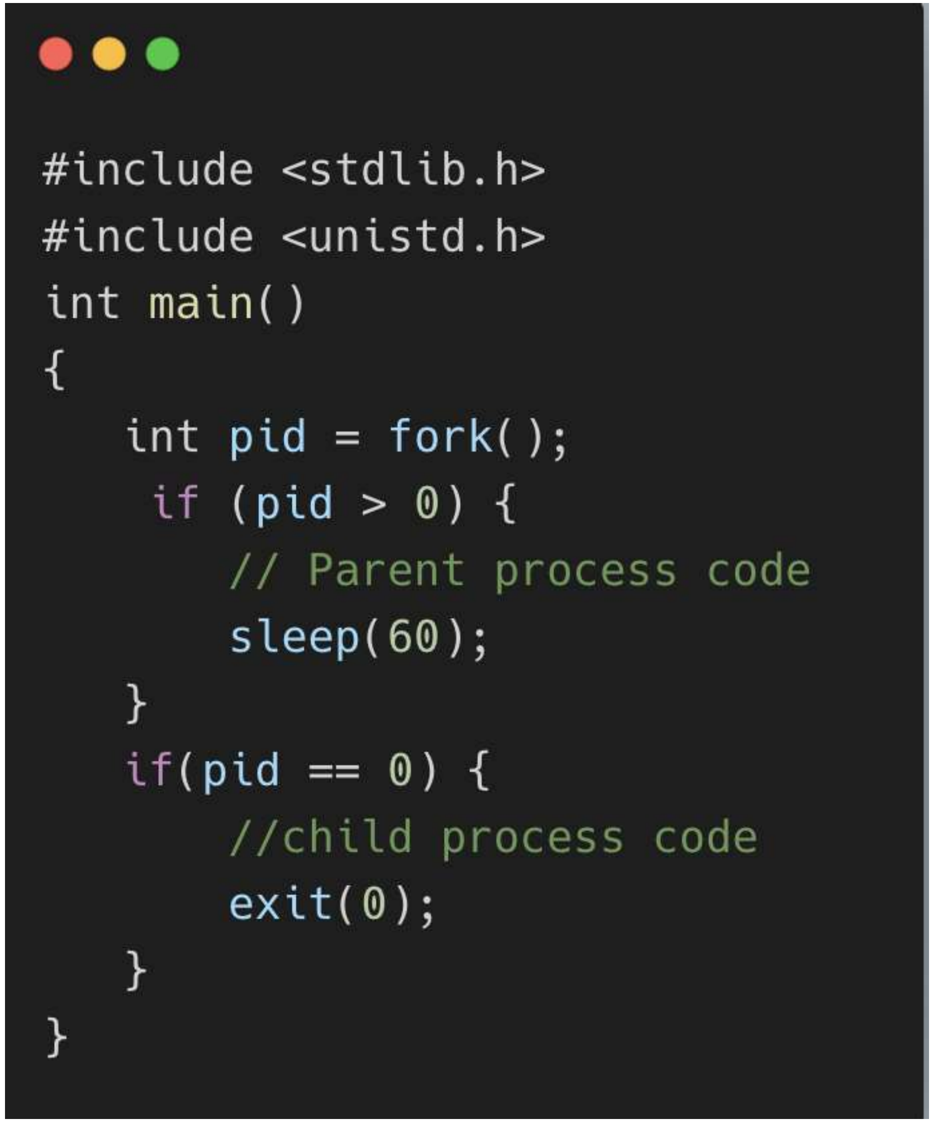
```
while(1) {
        fork();
}
```

- Fork bomb
- Number of PIDs is limited in the operating system. Once all PIDs are taken by the zombie process, no new process can be created.
- Since no new process can be created, one cannot even run a command on the command line.
- The only option to recover from a fork() bomb is to reboot the system.

Orphan Process

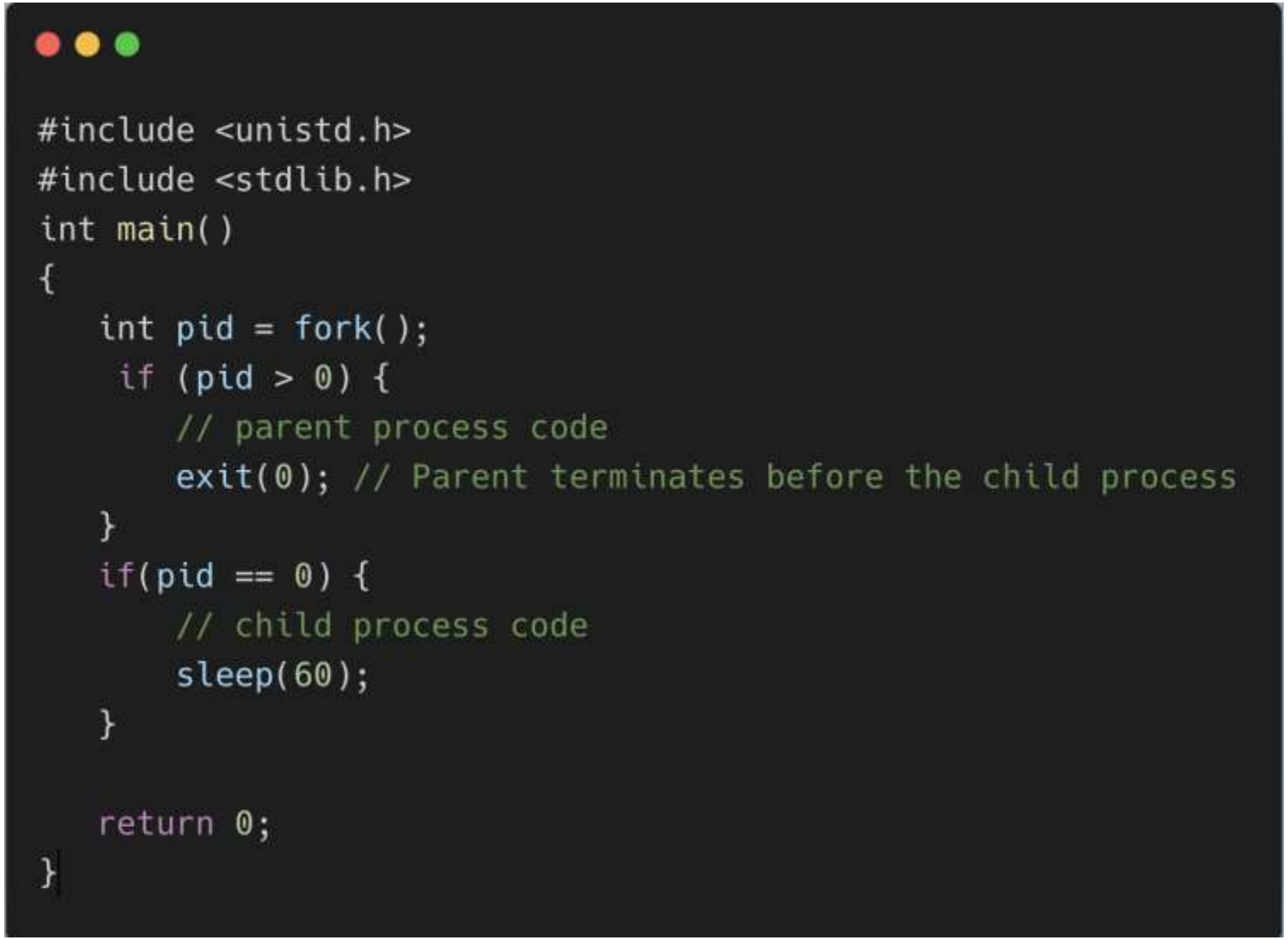
- A process which doesn’t have a parent process is called an orphan process.
- This may happen if the parent process terminates before child completes.
- What happens to the orphan process?
- Orphan process gets reparented and the new parent is the INIT process in most cases.
- The new parent then waits for the child to complete and then asks the kernel to clean the PCB.

ZOMBIE PROCESS PROGRAM



```c
#include <stdlib.h>
#include <unistd.h>
int main()
{
    int pid = fork();
     if (pid > 0) {
        // Parent process code
        sleep(60);
    }
    if(pid == 0) {
        //child process code
        exit(0);
    }
}
```

ORPHAN PROCESS PROGRAM



```
#include <unistd.h>
#include <stdlib.h>
int main()
{
   int pid = fork();
    if (pid > 0) {
       // parent process code
       exit(0); // Parent terminates before the child process
   }
   if(pid == 0) {
       // child process code
       sleep(60);
   }

   return 0;
}
```

References

- https://shivammitra.com/operating%20system/zombie-and-orphan-process-in-opearting-system
- Zombie and orphan process video - https://youtu.be/L3YQDUuDjoo
- https://shivammitra.com/operating%20system/fork-exec-wait-in-operating-system/