CS214-system programming

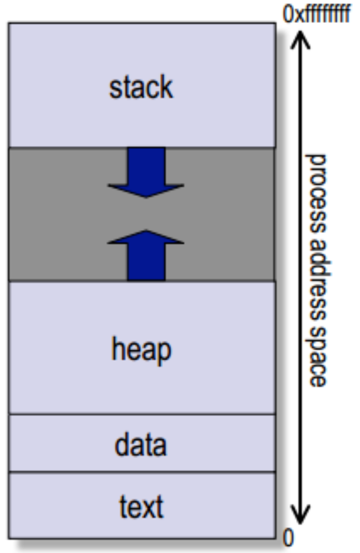
- Section 03/08 recitation 06

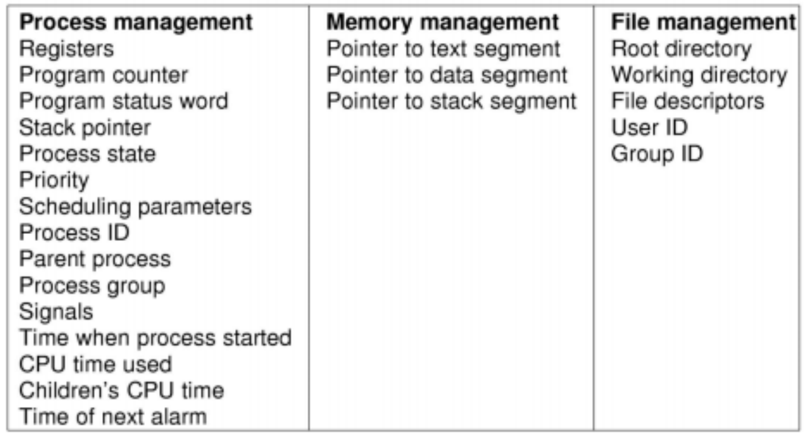
Yunhe Gao
yg397@scarletmail.rutgers.edu

Processes

- A process is an instance of a computer program that is being executed, it contains the program code and its current activity

- Each process has a **Process Control Block (PCB)** to describe its current execution state and some resources it is now processing
    - Identification information (e.g. parent process, user, process ID)
    - Execution contexts such as separate threads of execution
    - Address space (virtual memory)
    - I/O state (file handles, network communication endpoints, etc.)
    - etc.
- PCBs are stored as entries in a process table





Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Fields of a Process Control Block

Fork

```
int pid = fork();
```

- fork() is a system call that creates another process.
- The new process is called the **child** and the creating process is called the **parent**.
- The child is a copy of the parent process except for identification and scheduling state.
  - SAME: They share process image and process control structure
    - That is, they share the same code, variables, file descriptors, signal disposition (reaction to signals)
  - DIFFERENT: PID, parent PID (PPID), and address space
    - Parent and child run in two different address spaces. There is **no memory sharing** by default.

Fork

```
int pid = fork();
```

- fork() is called once, but returns twice.
- Once for the parent process, and once for the child.

- Returns an integer.

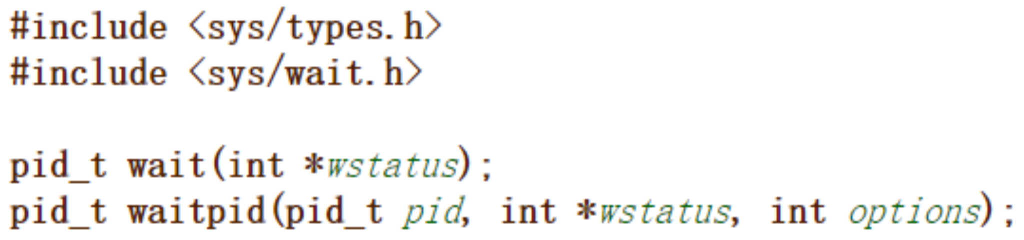
• negative	-- Error.
• zero	-- Zero is returned to the child process.
• positive	-- The PID of child process is returned to the parent process

Wait

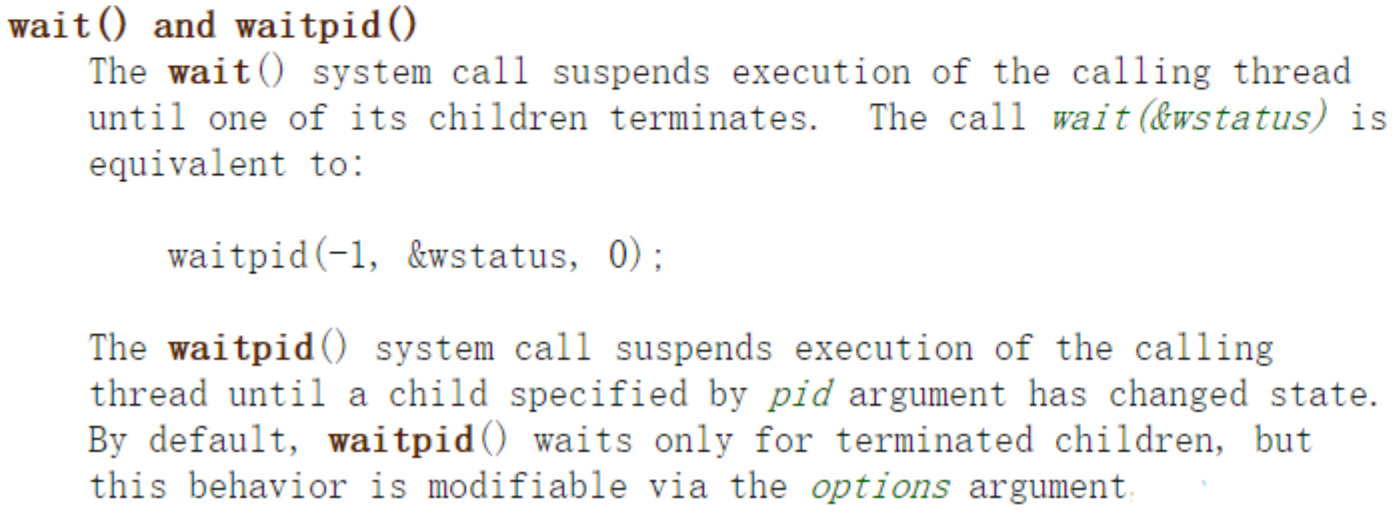
When a process terminates execution, the OS releases most of the resources and information related to that process.

- It does keep data about resource utilization and termination status in case the parent process wants to know.
- The process still takes up space in the OS's process table.

After the parent receives the information, the OS releases the rest of the resources tied to the terminated process and removes its pid from the resource table.



```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *wstatus);
pid_t waitpid(pid_t pid, int *wstatus, int options);
```



wait() and waitpid()

The **wait**() system call suspends execution of the calling thread until one of its children terminates. The call *wait(&wstatus)* is equivalent to:

```
    waitpid(-1, &wstatus, 0);
```

The **waitpid**() system call suspends execution of the calling thread until a child specified by *pid* argument has changed state. By default, **waitpid**() waits only for terminated children, but this behavior is modifiable via the *options* argument.

Fork + wait

```
int pid = fork();

if(pid == 0){
        printf(“This is the child process.\n”);
        exec(something);       // call a program different from the parent program
        exit(EXIT_FAILURE);
        }
else{
        printf(“This is the parent process.\n”);
        int status;
        int pid = wait(&status); // write the exit information into status
        printf(“Got status: %d\n”, WEXITSTATUS(status));  //use macro to inspect the return
                                                              //status of child process
        }
```

exec

- The exec() family of functions **replaces** the current process image with a new process image. It loads the program into the current process space and runs it from the entry point.
- `int execl(const char *path, const char *arg, ...);`
- `int execv(const char *path, char *const argv[]);`
- Args:
  - The path to the program to execute
  - The arguments to the program
- exec only returns when error occurs
  - If success, the process is replaced by a new one. There is no place to return.
- Much of other process information is preserved
  - Open files

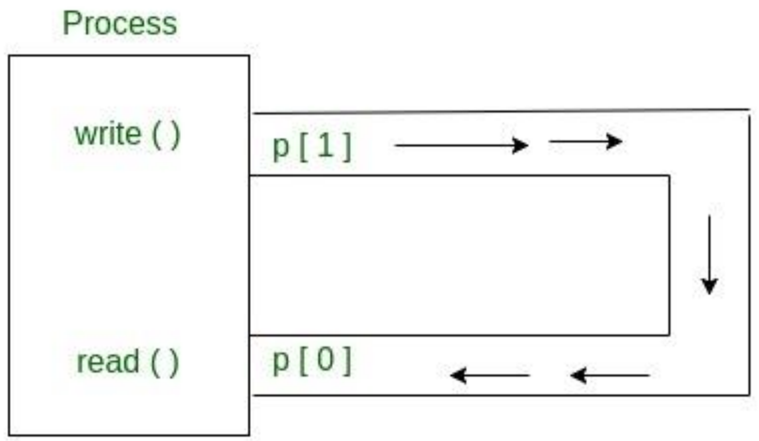
Pipe

- A pipe is a connection between two processes, which is useful for communication between related processes (inter-process communication)

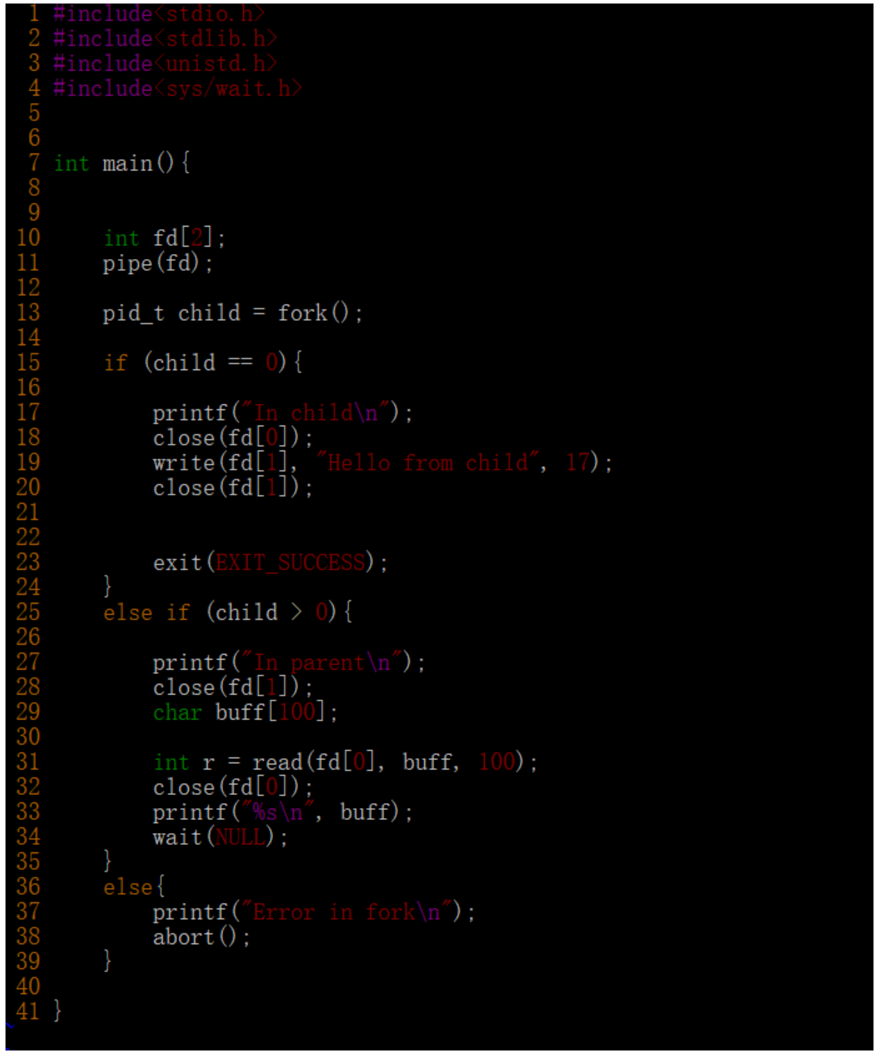
```
int pipe(int fds[2]);
Parameters :
fd[0] will be the fd(file descriptor) for the read end of pipe.
fd[1] will be the fd for the write end of pipe.
Returns :
0 on Success.
-1 on error.
```

Pipe

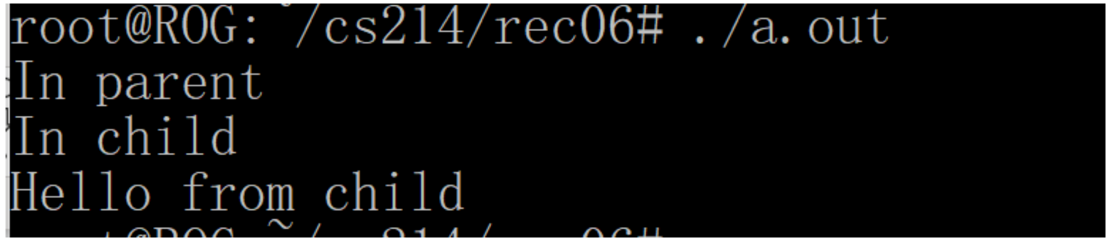
- Pipe is one-way communication only i.e we can use a pipe such that One process write to the pipe, and the other process reads from the pipe. It opens a pipe, which is an area of main memory that is treated as a *"virtual file"*.
- The pipe can be used by the creating process, as well as all its child processes, for reading and writing. One process can write to this "virtual file" or pipe and another related process can read from it.
- If a process tries to read before something is written to the pipe, the process is suspended until something is written.
- The pipe system call finds the first two available positions in the process's open file table and allocates them for the read and write ends of the pipe.



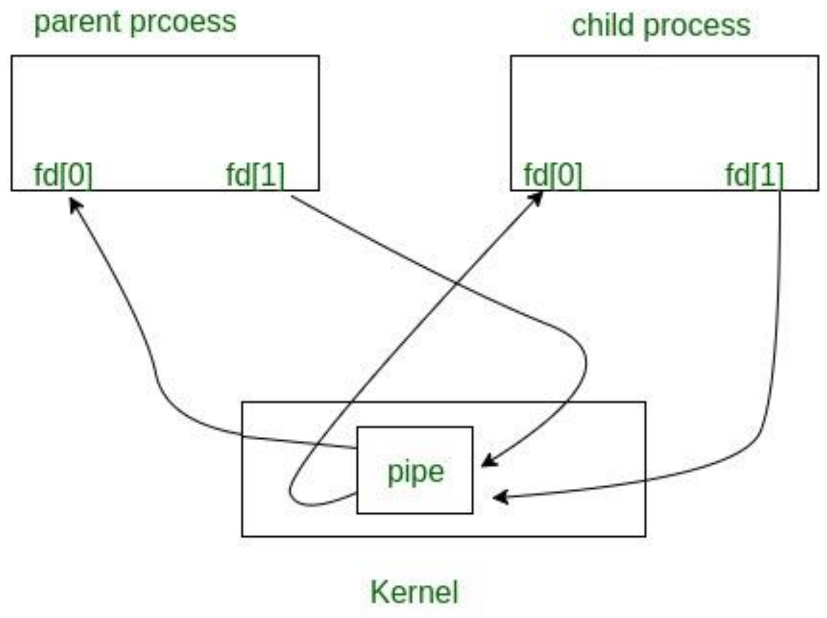
Pipe + fork



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


int main(){


    int fd[2];
    pipe(fd);

    pid_t child = fork();

    if (child == 0){

        printf("In child\n");
        close(fd[0]);
        write(fd[1], "Hello from child", 17);
        close(fd[1]);


        exit(EXIT_SUCCESS);
    }
    else if (child > 0){

        printf("In parent\n");
        close(fd[1]);
        char buff[100];

        int r = read(fd[0], buff, 100);
        close(fd[0]);
        printf("%s\n", buff);
        wait(NULL);
    }
    else{
        printf("Error in fork\n");
        abort();
    }

}
```



```
root@ROG:~/cs214/rec06# ./a.out
In parent
In child
Hello from child
```



Thanks