Operating Systems (CS3000)

Lecture – 11

(fork() System Call)

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY,
DESIGN AND MANUFACTURING,
KANCHEEPURAM

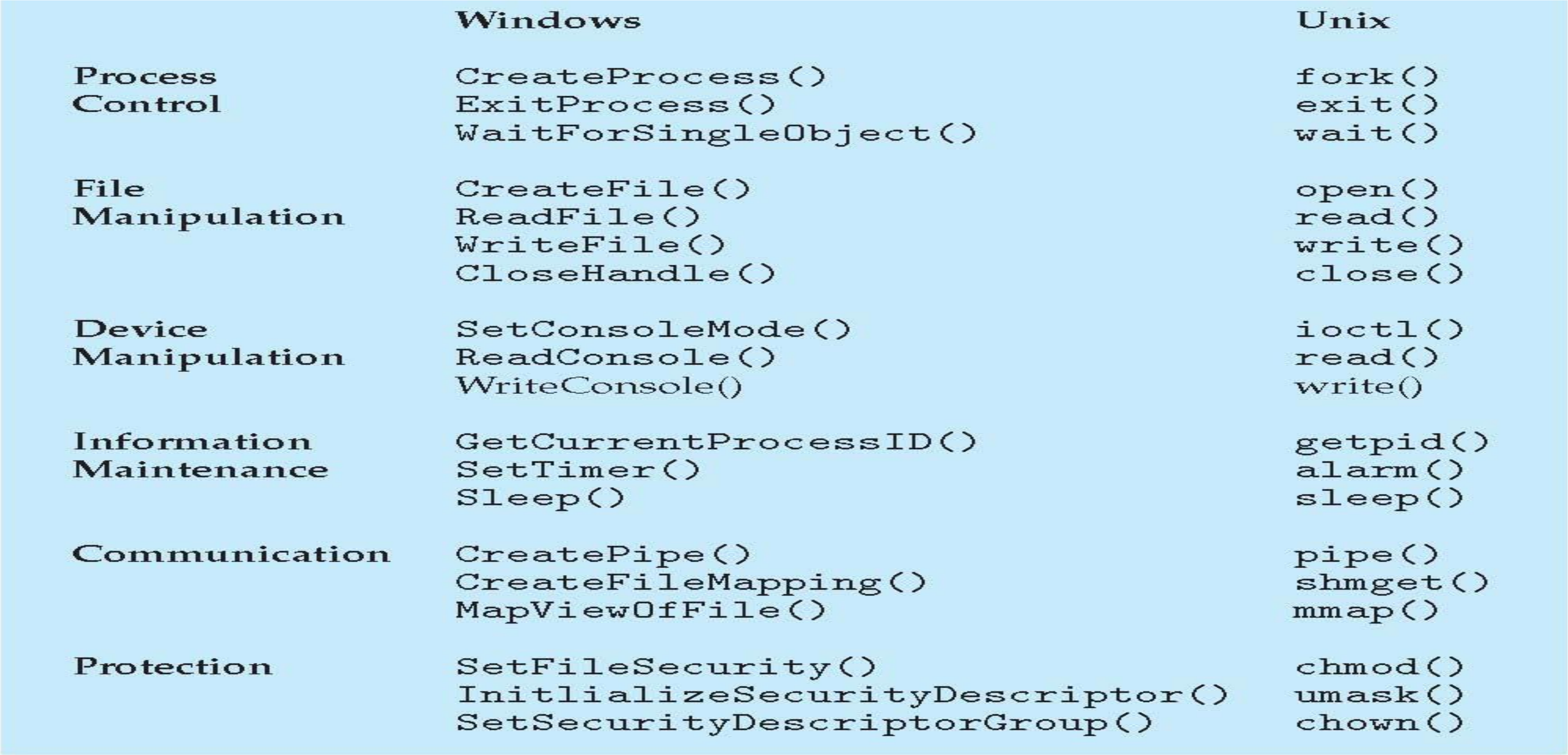
Dr. Jaishree Mayank

Assistant Professor

Department of Computer Sc. and Engg.

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY,
DESIGN AND MANUFACTURING,
KANCHEEPURAM

Examples of Windows and Unix System Calls



	Windows	Unix

Process Control	CreateProcess()
	ExitProcess()
	WaitForSingleObject()
		fork()
		exit()
		wait()

File Manipulation	CreateFile()
	ReadFile()
	WriteFile()
	CloseHandle()
		open()
		read()
		write()
		close()

Device Manipulation	SetConsoleMode()
	ReadConsole()
	WriteConsole()
		ioctl()
		read()
		write()

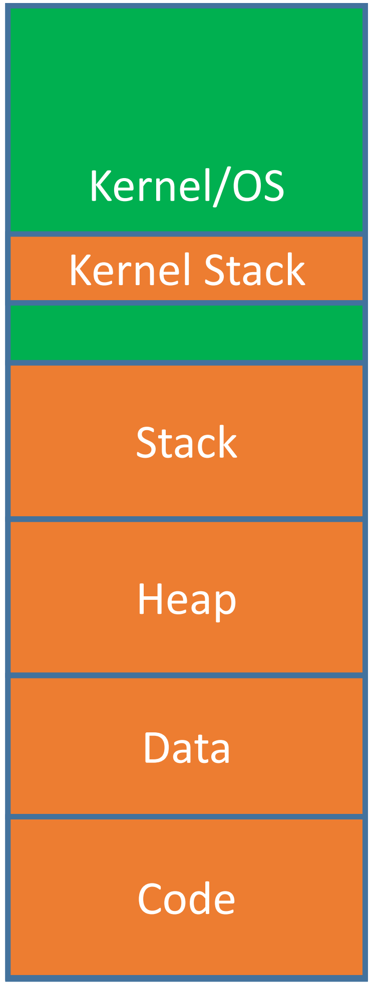
Information Maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

Communication	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shmget()
		mmap()

Protection	SetFileSecurity()
	InitlializeSecurityDescriptor()
	SetSecurityDescriptorGroup()
		chmod()
		umask()
		chown()

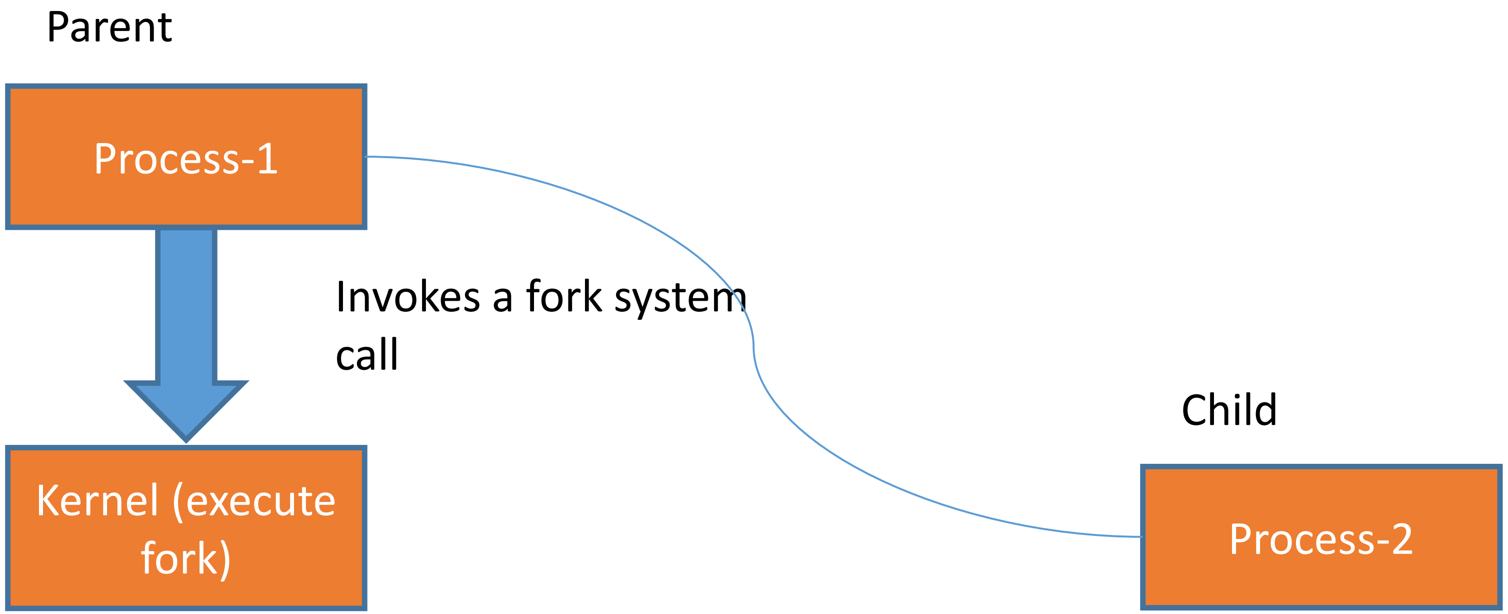
What Metadata of a Process Kernel Stores?

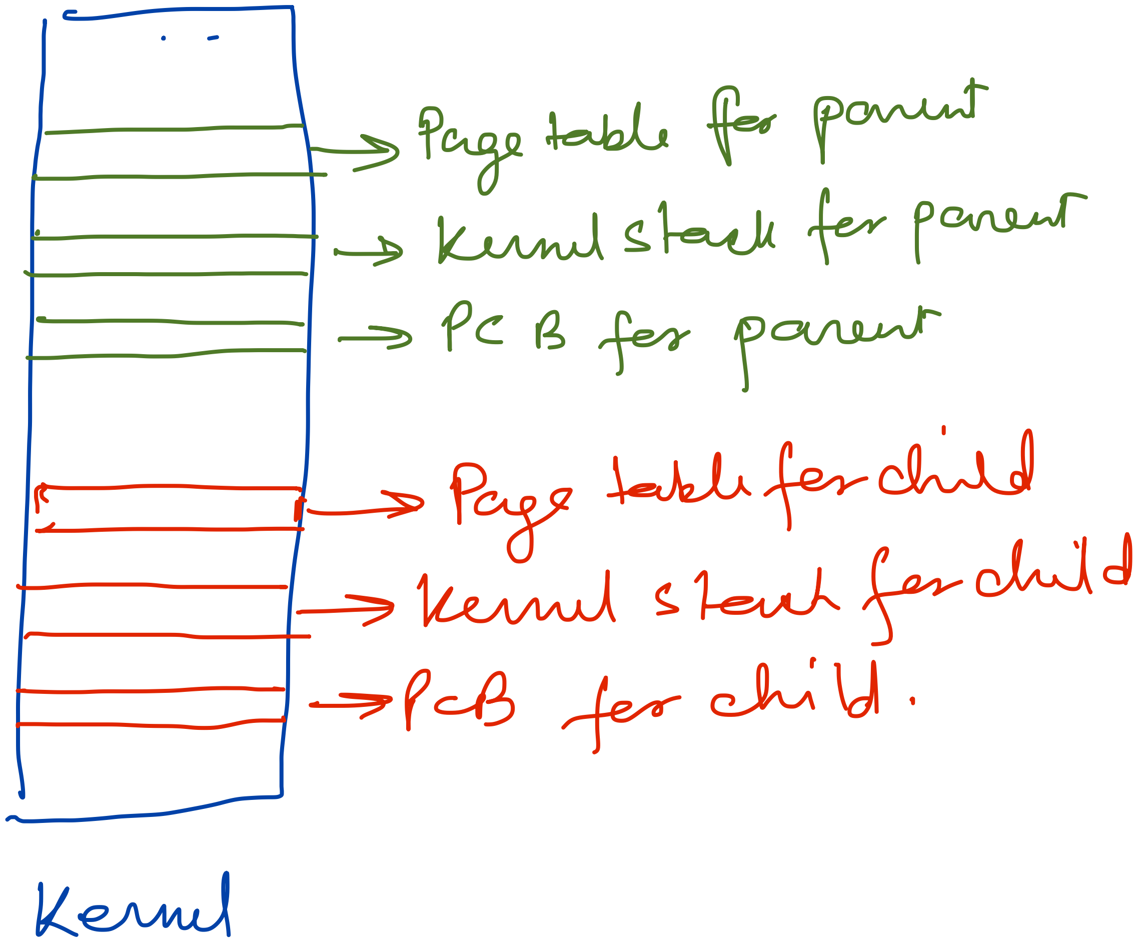
- PCB
- Kernel Stack for User Process
  - During System Calls
- Page Table for that User Process



Creating a Process by Cloning

- fork()
  - Child Process is duplicate of parent process
  - PID → Parent process is Child’s PID
  - PID → Child process is 0





- * fork system call creates exact replica of parent process
- * Creates PT, kernel stack & PCB for child process.
  - → set state to New for child proc
  - → Other information same as parent
- * After completion of fork() it change the state of child process to READY.
- * Return child id to parent
  Return 0 to child.

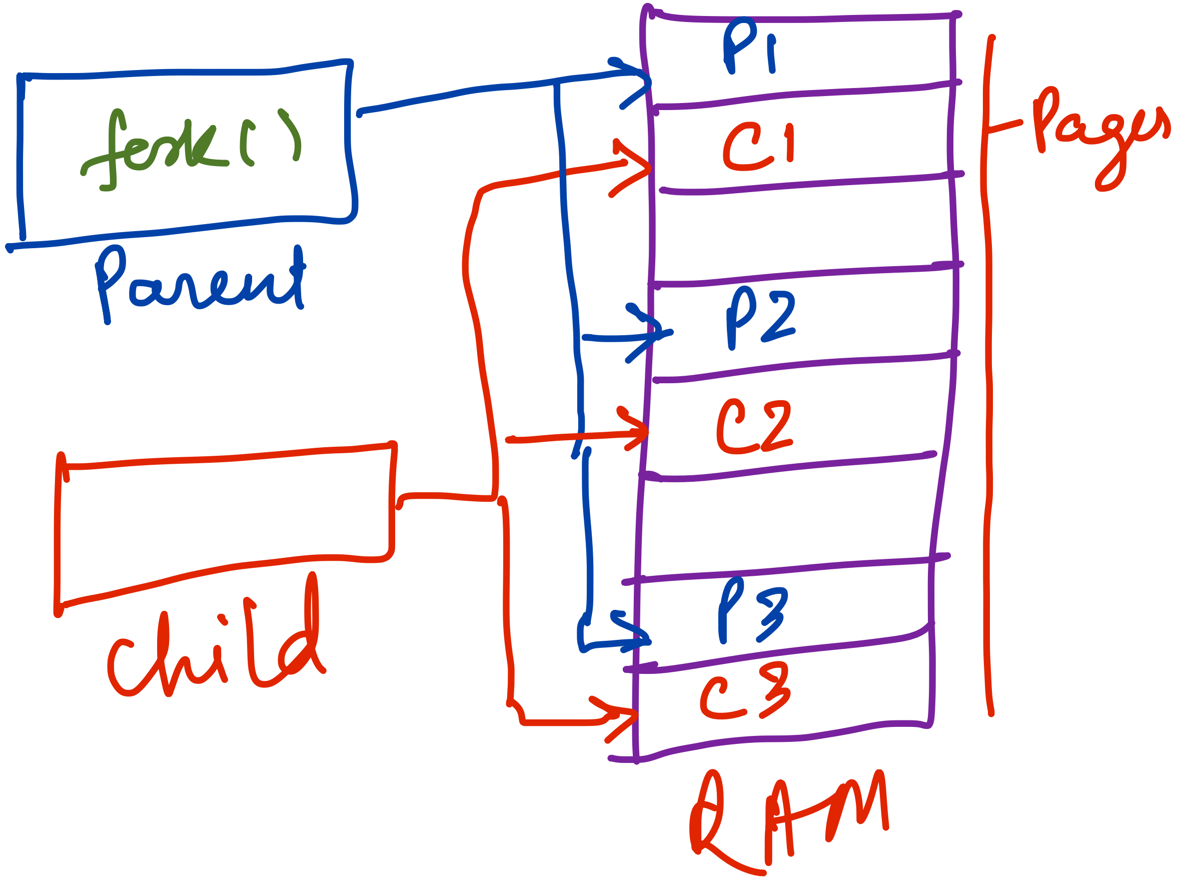
Two ways to allocate space for child process

Case 1

* different space for child, so the page table entries will also be different.

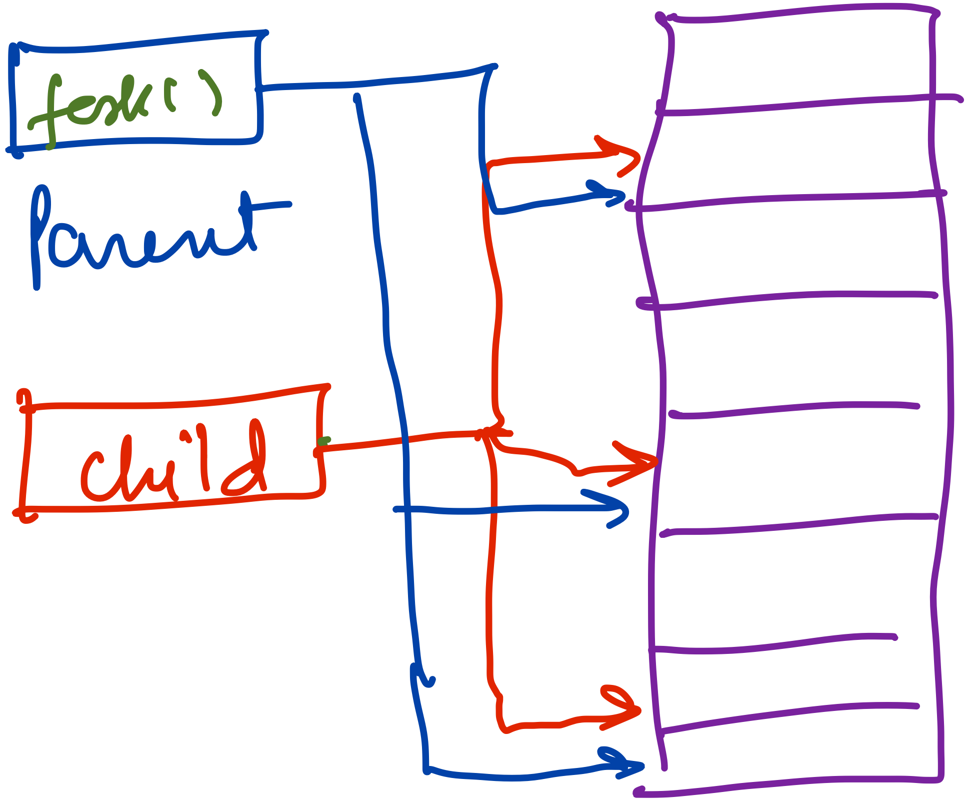
Case 2

* Share the same physical space as parent, so the page table entries will be same.



Case 1

Separate, duplicate Non-shared pages



Case 2

Separate, duplicate, shared pages

fork( )

Parent

```
{
   pid_t pid;
   pid=fork();

   if (pid<0)
      printf("error in fork \n");
   else if (pid==0)
   {
      fork();
      printf("child print \n");
   }
   else if (pid>0)
      printf("Parent Print \n");
   printf("Main Print \n");
   return 0;
}
```

child

```
{
   pid_t pid;


   if (pid<0)
      printf("error in fork \n");
   else if (pid==0)
   {
      fork();
      printf("child print \n");
   }
   else if (pid>0)
      printf("Parent Print \n");
   printf("Main Print \n");
   return 0;
}
```

Grandchild

```
{
   pid_t pid;


   if (pid<0)
      printf("error in fork \n");
   else if (pid==0)
   {

      printf("child print \n");
   }
   else if (pid>0)
      printf("Parent Print \n");
   printf("Main Print \n");
   return 0;
}
```

Output :-

from parent process { Parent print
Main print

child process { child print
Main print

{ child print
Main print

→ Grandchild

Thank You

Any Questions?