Process

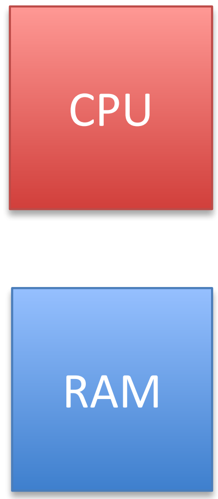
Md. Asifur Rahman
Lecturer
Dept. of CSE, RUET

What is Process ?

- A **process** is a program in execution.
- Process is the OS's abstraction for execution.

Process consists of (At least)

- CPU state
- An address space.
- OS resources.



Program Vs Process

Program:

- Passive entity for a process
- Many users can run the same program

  Each process has its own address space, i.e., even though program has single set of variable names, each process will have different values

Process:

- Active entity of a program.
- A program can invoke more than one process.

Process Management

Fundamental task of any OS.

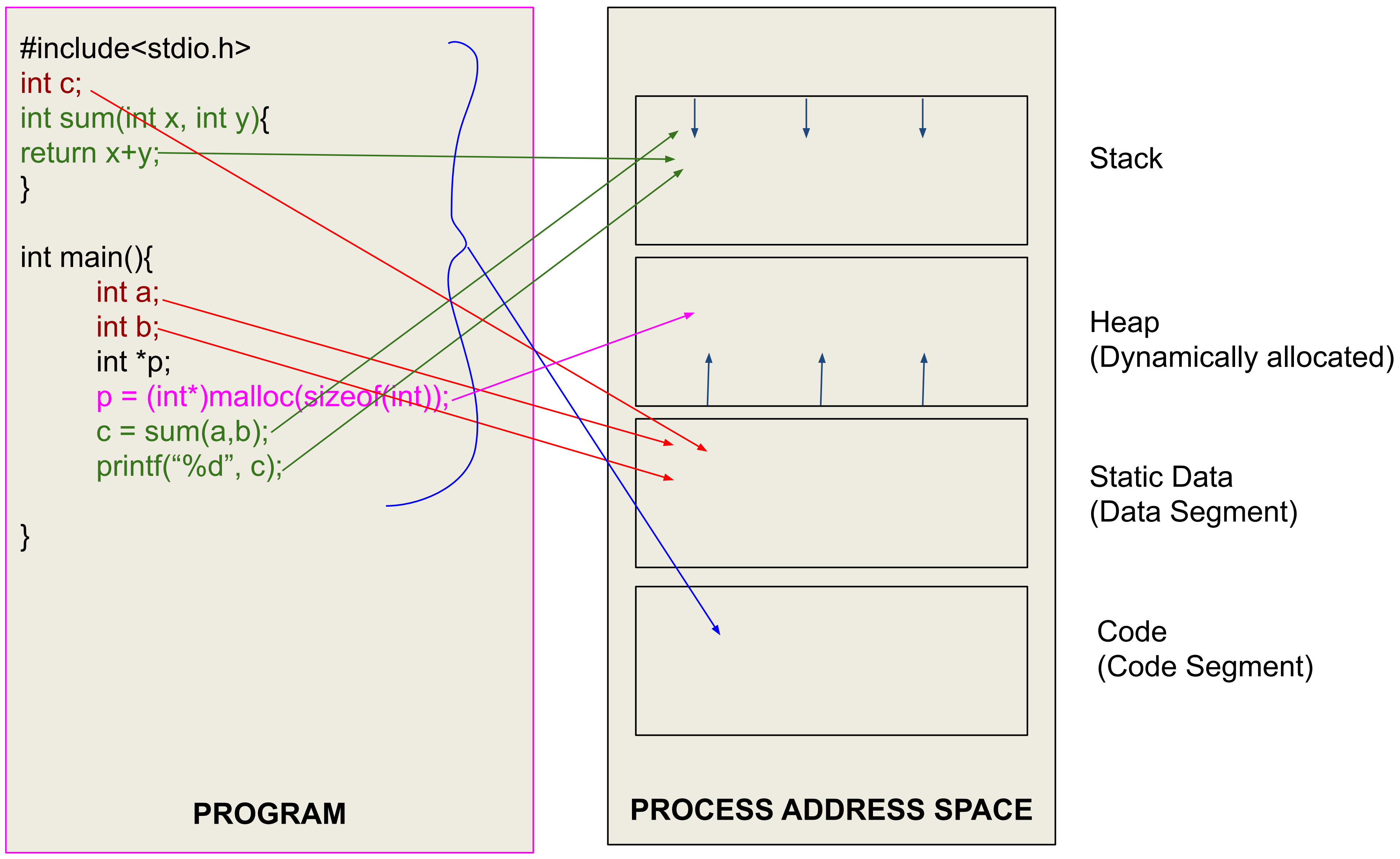
OS must :

- Allocate resource.
- Protect resource from other process.
- Enable Synchronization.

How?

- Allocate process to its own address space
- Describing process state.
- Manage process using a data structure - PCB

Process Address Space



Process State

A process may be in one of the following states:

- New: The process being created.
- Running: Instruction are being executed
- Waiting: The process is waiting for some event to occur.
- Ready: The process in waiting to be assigned to a processor.
- Terminated: The process has finished execution.

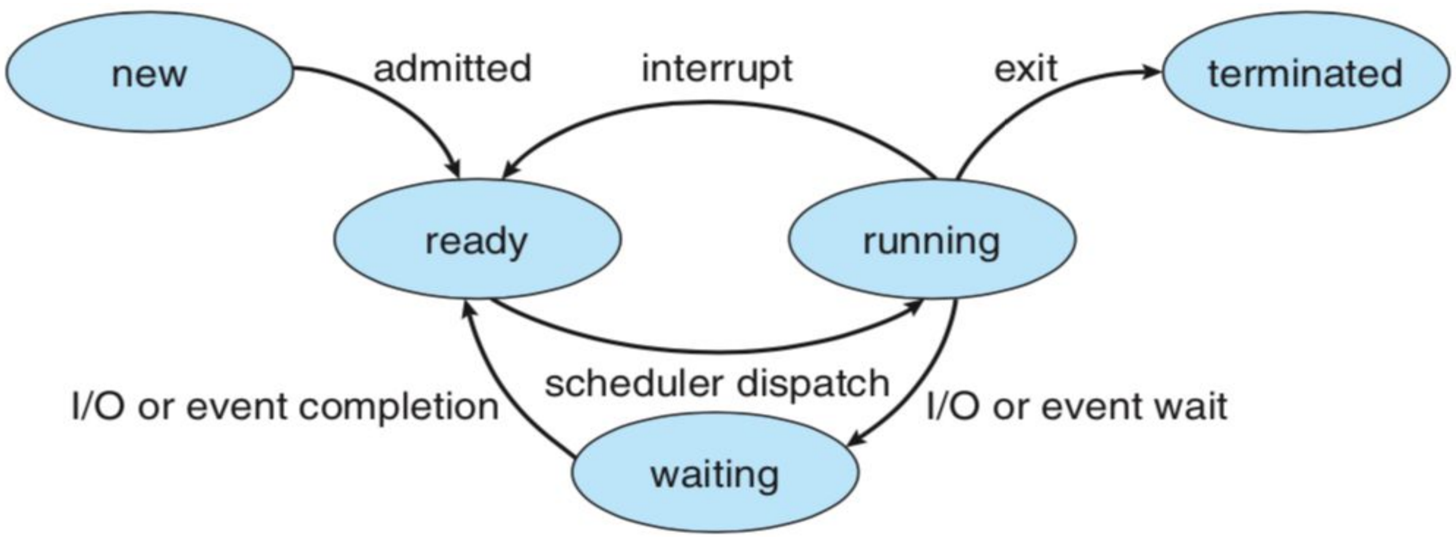


Figure 3.2 Diagram of process state.

Process Control Block

- **Process state**. The state may be new, ready, running, waiting, halted, and so on.
- **Program counter**. The counter indicates the address of the next instruction to be executed for this process.
- **CPU registers**. The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward (Figure 3.4).
- **CPU-scheduling information**. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters. (Chapter 6 describes process scheduling.)
- **Memory-management information**. This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system (Chapter 8).

Process Control Block

- **Accounting information**. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- **I/O status information**. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

Process ID
Process State
Program Counter
CPU Registers
Scheduling information
Memory management information
Accounting information
I/O status information

Context Switch

Context : PCB status of a process.
Switch: Alternate between running process and executing process by saving the current context of running process in PCB and then loading the context of waiting process on the CPU.

OS switches between process / Thread in the following manner:

- Interrupt the running process or thread.
- Store its CPU status as context (in PCB).
- Run **OS scheduler** to decide what to do next.
- Flush the cache if required.
- Load the context (from PCB) of the chosen next thread/ process in CPU.
- Begin executing that thread/ process.

Context Switch

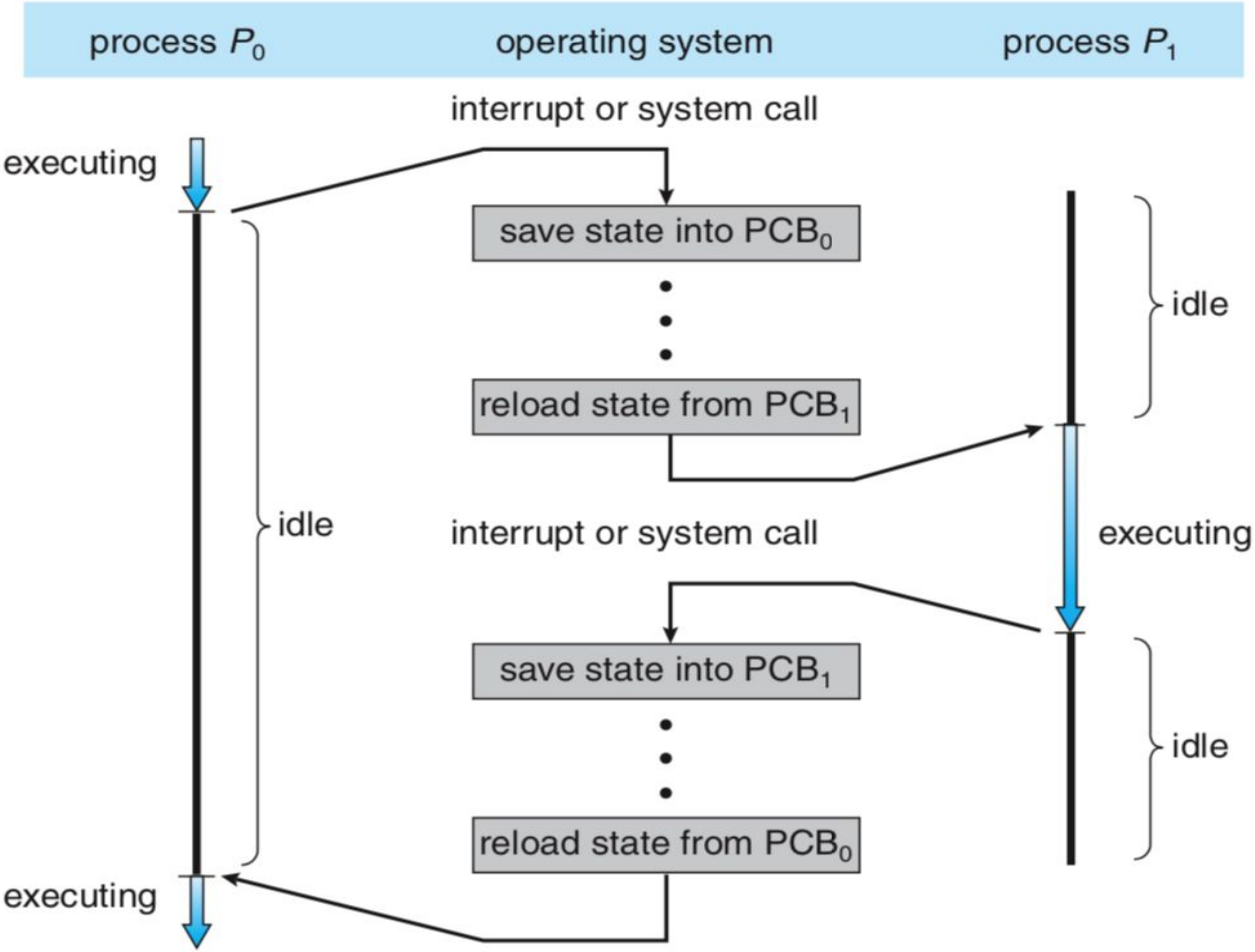
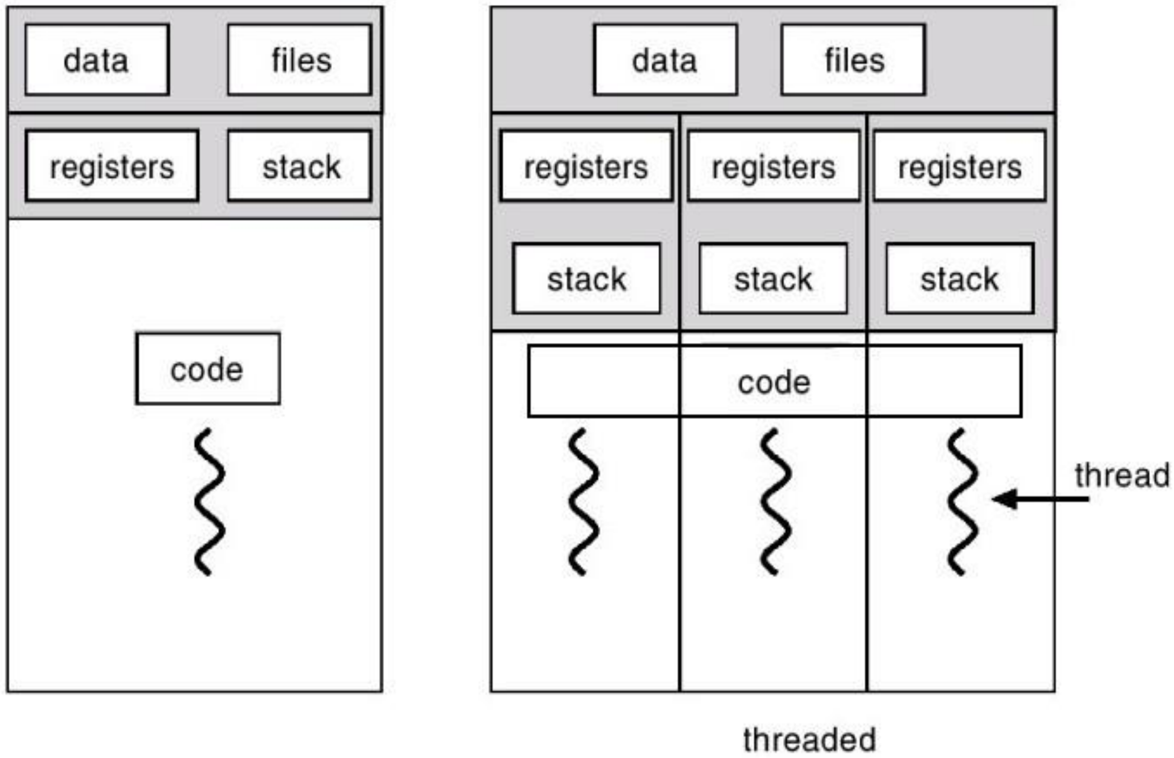


Figure 3.4 Diagram showing CPU switch from process to process.

Thread

Thread is a single sequential execution stream within a process. It is a basic unit of CPU utilization that comprises a thread ID, program counter, a register set and a stack.



Thread

Case study into the application of thread:

- A web server accepts client requests for web pages, images, sound, and so forth. A busy web server may have several (perhaps thousands of) clients concurrently accessing it. If the web server ran as a traditional single-threaded process, it would be able to service only one client at a time, and a client might have to wait a very long time for its request to be serviced.
- The above problem can be alternatively solve by creating a new process every time a new client request, however process creation is time consuming and resource intensive.

Process Creation

A parent process can create a new process called child process using create-process system call.

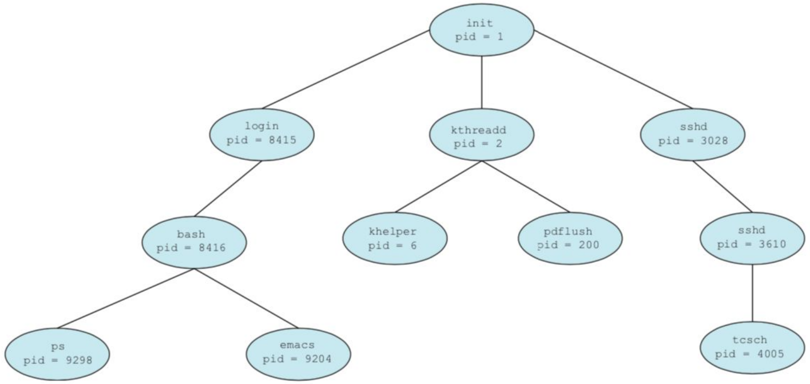


Figure 3.8 A tree of processes on a typical Linux system.

When a process creates a new process two possibilities exists in term of execution:

- The parent continues to execute concurrently with its children.
- The parent waits until some or all of its children have terminated.

In terms of the address space there exists two possibilities:

- The child process is duplicate of the parent process that is it has the same program and data as the parent but different address space. **(fork)**
- The child process has different address space and a new program is loaded into it.**(exec)**

Process Creation

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

int main(int argc, char *argv[])
{
   int a = 100;
   int id1 = fork();
   int id2 = fork();
   printf("%d\n",a);
   return 0;
}
```

fork() creates a child process that is a clone of the parent process.

Which means that the child has the same code segment as the parents since it has the same page table as its parent process.

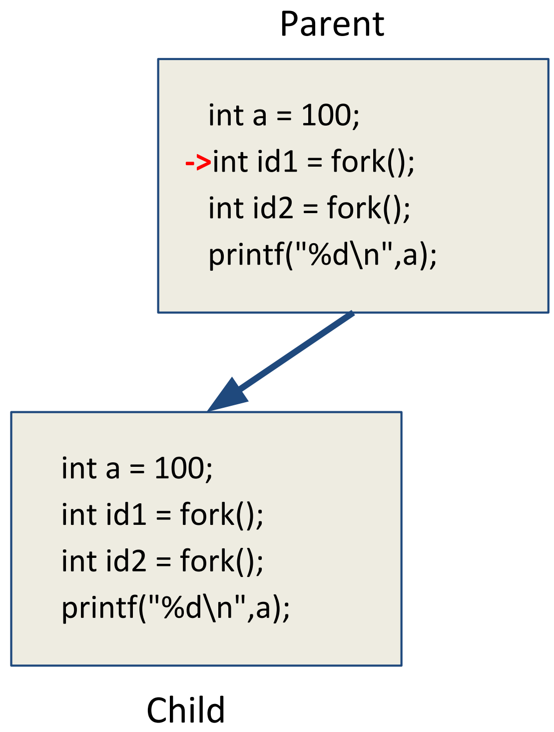
Process Creation

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

int main(int argc, char *argv[])
{
   int a = 100;
   int id1 = fork();
   int id2 = fork();
   printf("%d\n",a);
   return 0;
}
```

```
->int a = 100;
  int id1 = fork();
  int id2 = fork();
  printf("%d\n",a);
```

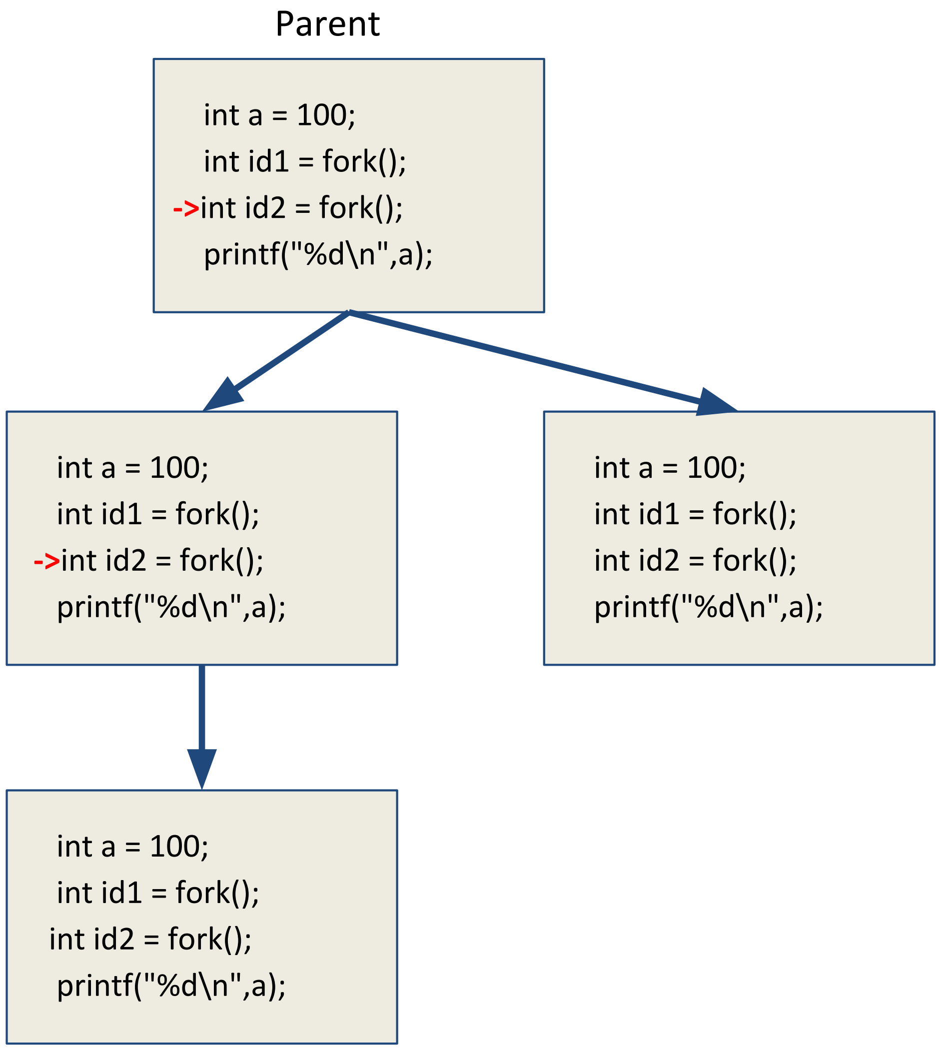
Process Creation

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

int main(int argc, char *argv[])
{
   int a = 100;
   int id1 = fork();
   int id2 = fork();
   printf("%d\n",a);
   return 0;
}
```



Process Creation

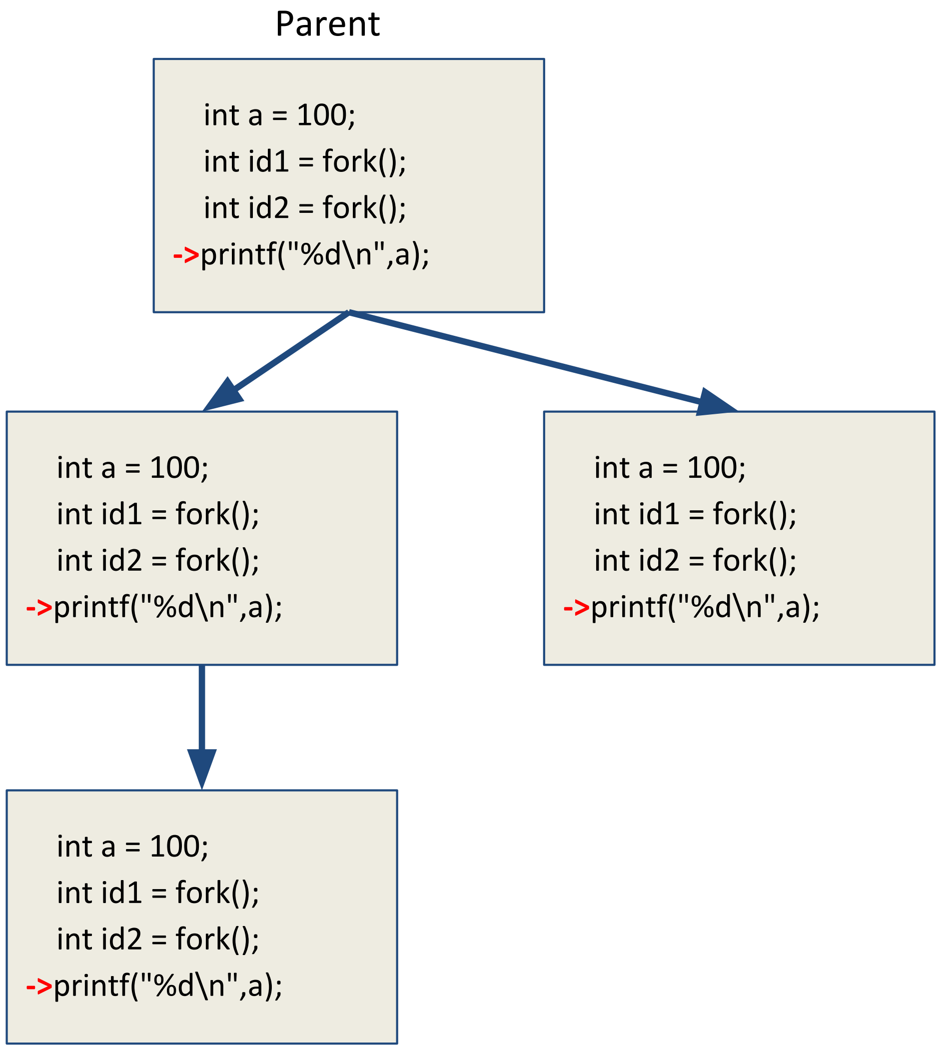
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

int main(int argc, char *argv[])
{
   int a = 100;
   int id1 = fork();
   int id2 = fork();
   printf("%d\n",a);
   return 0;
}
```



Process Creation

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

int main(int argc, char *argv[])
{
    int a = 100;
    int id1 = fork();
    int id2 = fork();
    printf("%d\n",a);
    return 0;
}
```

GUESS THE OUTPUT OF THIS PROGRAM



Reference

Book Ref:

1. Modern Operating System – by Tanenbaum

2.Operating System Concepts - by Abraham Silberschatz