STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

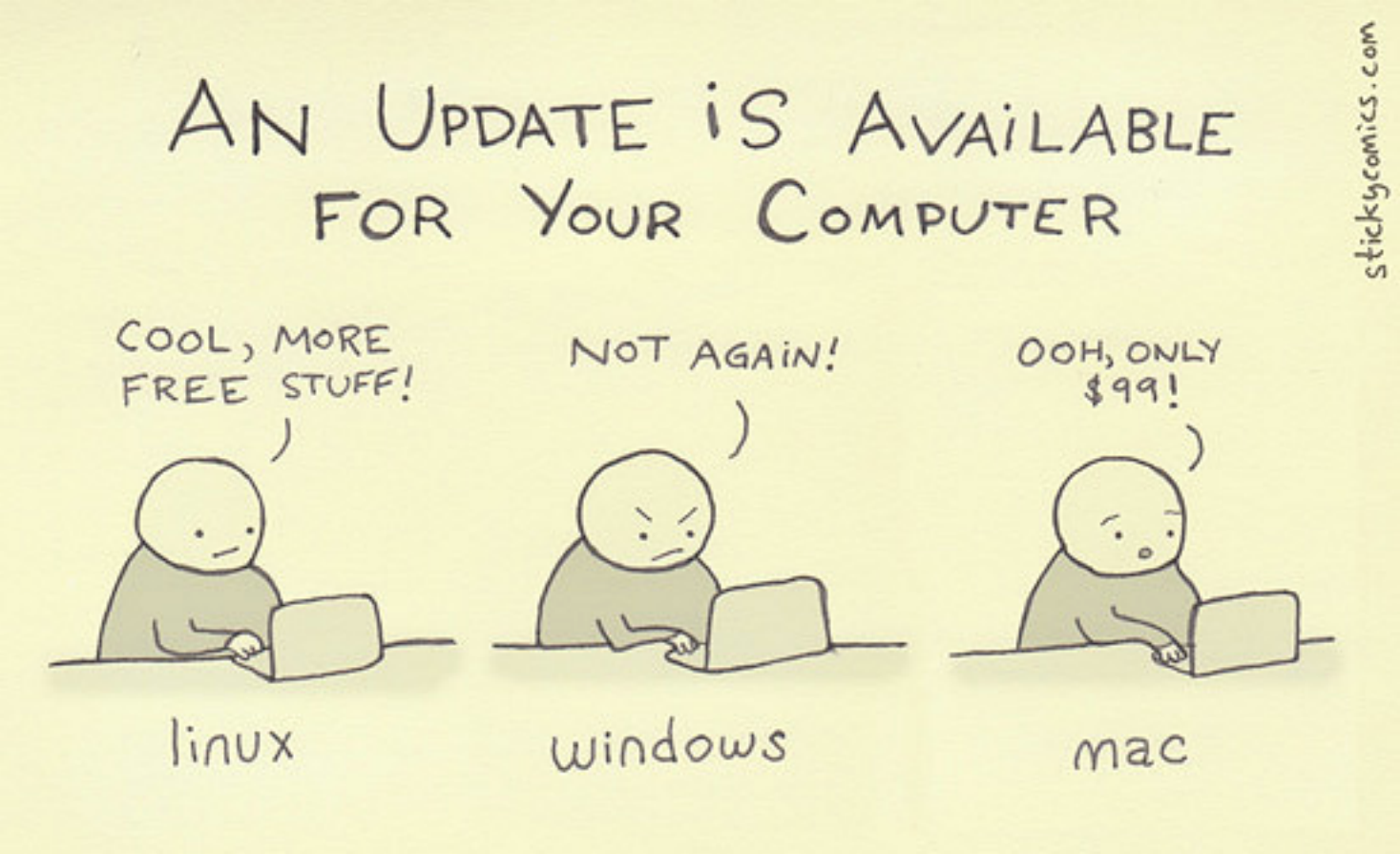
CS 492: Operating Systems

Processes (1)

Instructor: Iraklis Tsekourakis

Email: itsekour@stevens.edu

Processes


AN UPDATE IS AVAILABLE FOR YOUR COMPUTER
stickycomics.com
COOL, MORE FREE STUFF!
NOT AGAIN!
OOH, ONLY $99!
linux
windows
mac

Goals for Today

- Process
  - Concept
  - Operations on Processes
  - Process States
  - Process Implementation

What is Program?

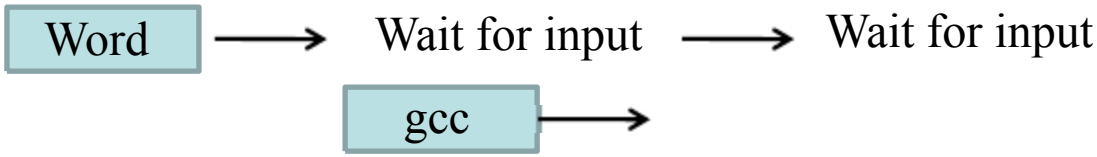
- A program
  - A set of instructions
  - NO need to be active

What is Process?

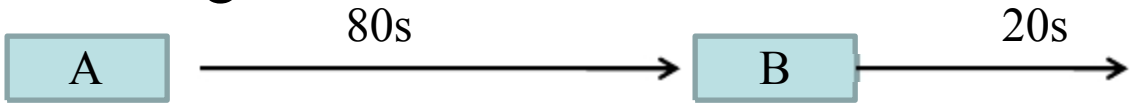
- The operating system creates processes from a program
- A process is an active program

Why Processes?

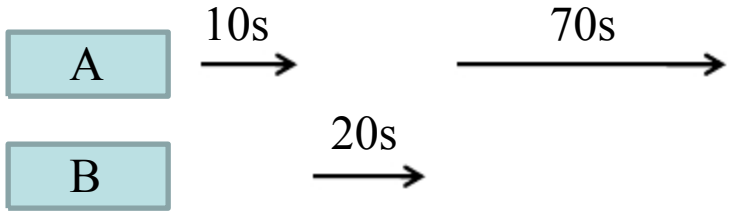
- Multiple processes increase CPU utilization
  - Overlap one process's computation with another's wait



- Multiple processes reduce latency
  - Running A then B: B takes 100s



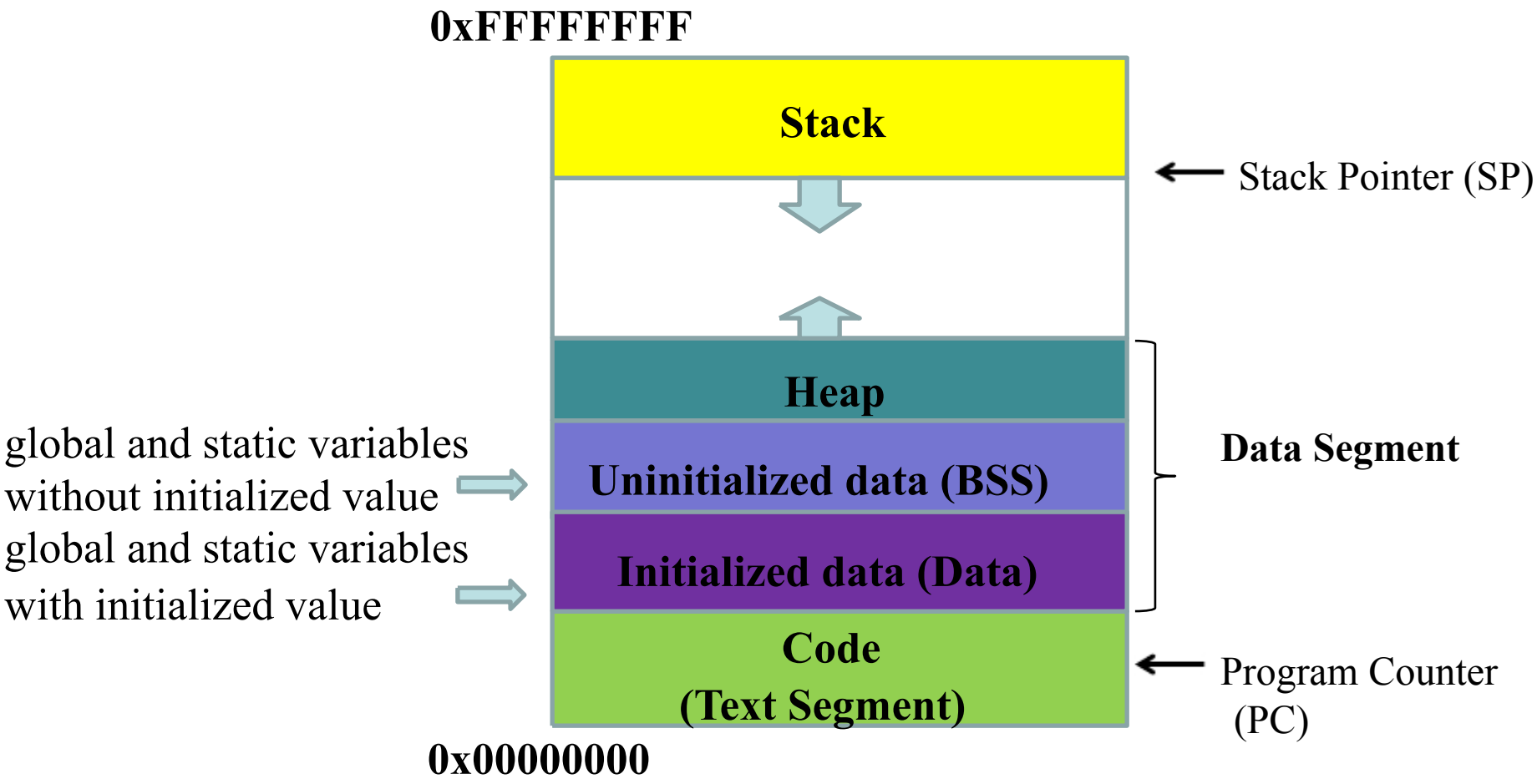
  - Running A and B simultaneously: B takes 30s



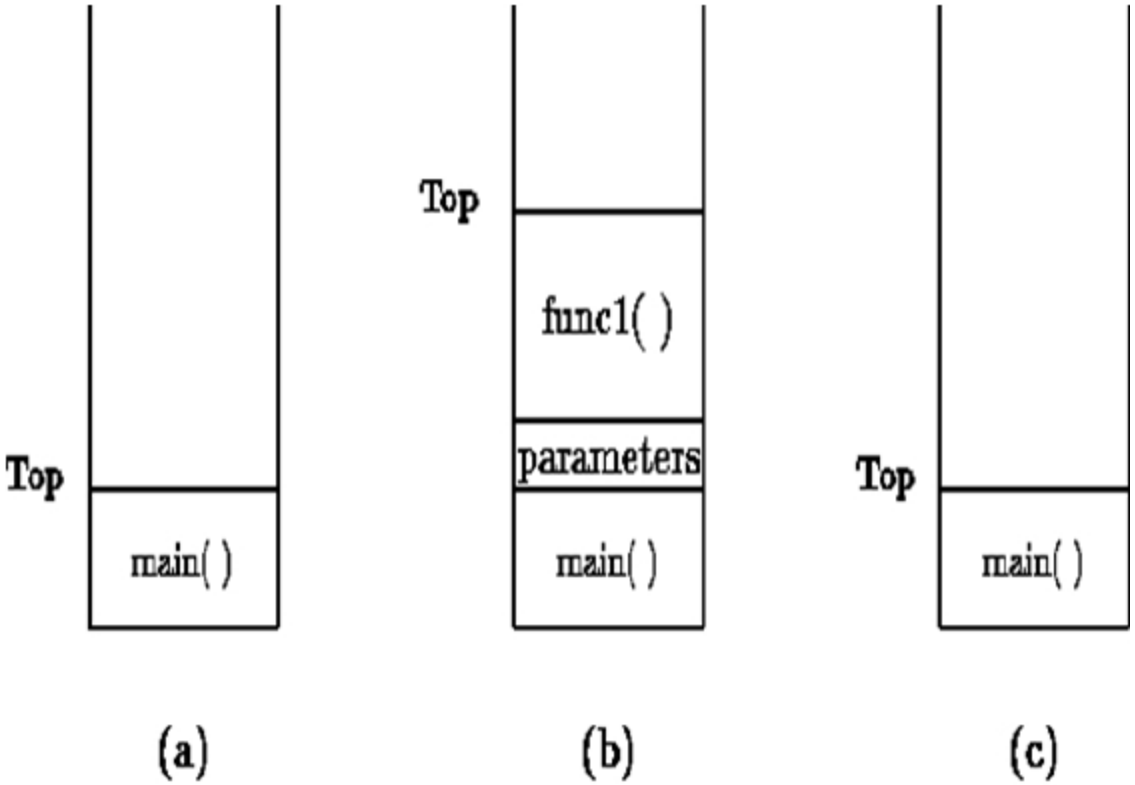
Components of Processes

- 3 components
  1. An executable program
  2. Associated data needed by the program
     - Program counter (PC)
     - Data segment
     - Text segment
     - Stack Pointer (SP)
  3. Execution context of the program

Diagram of A Single Process

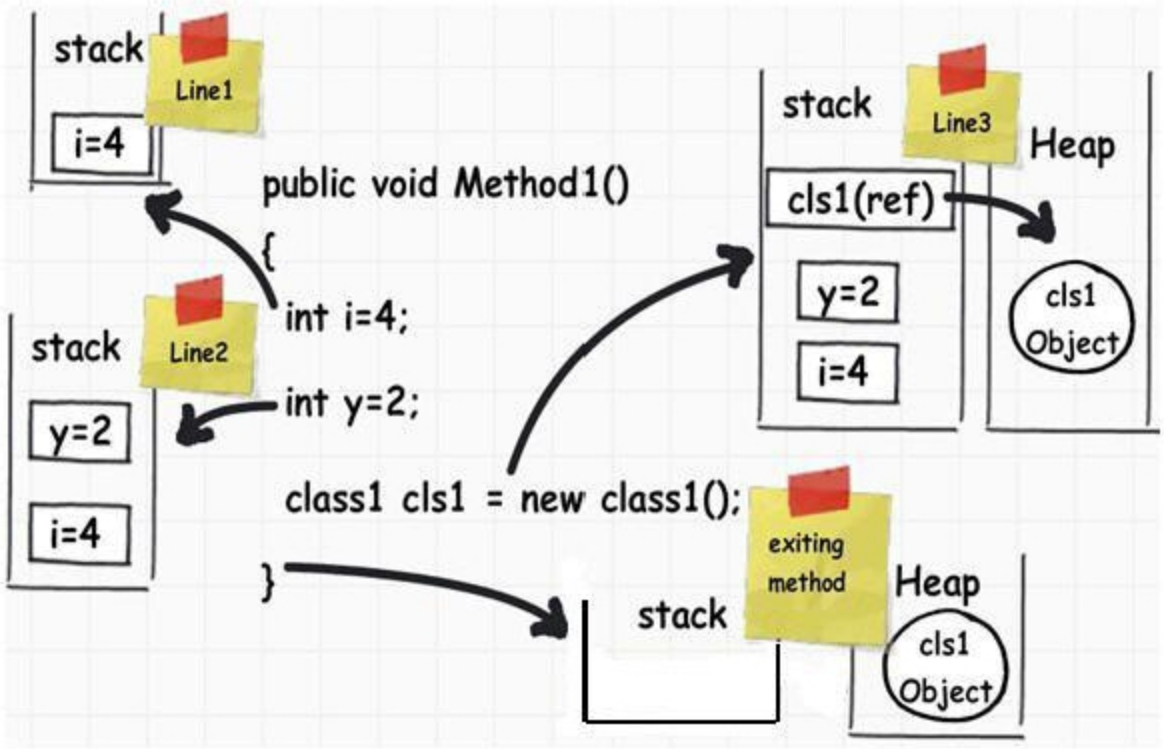


Stack VS. Heap



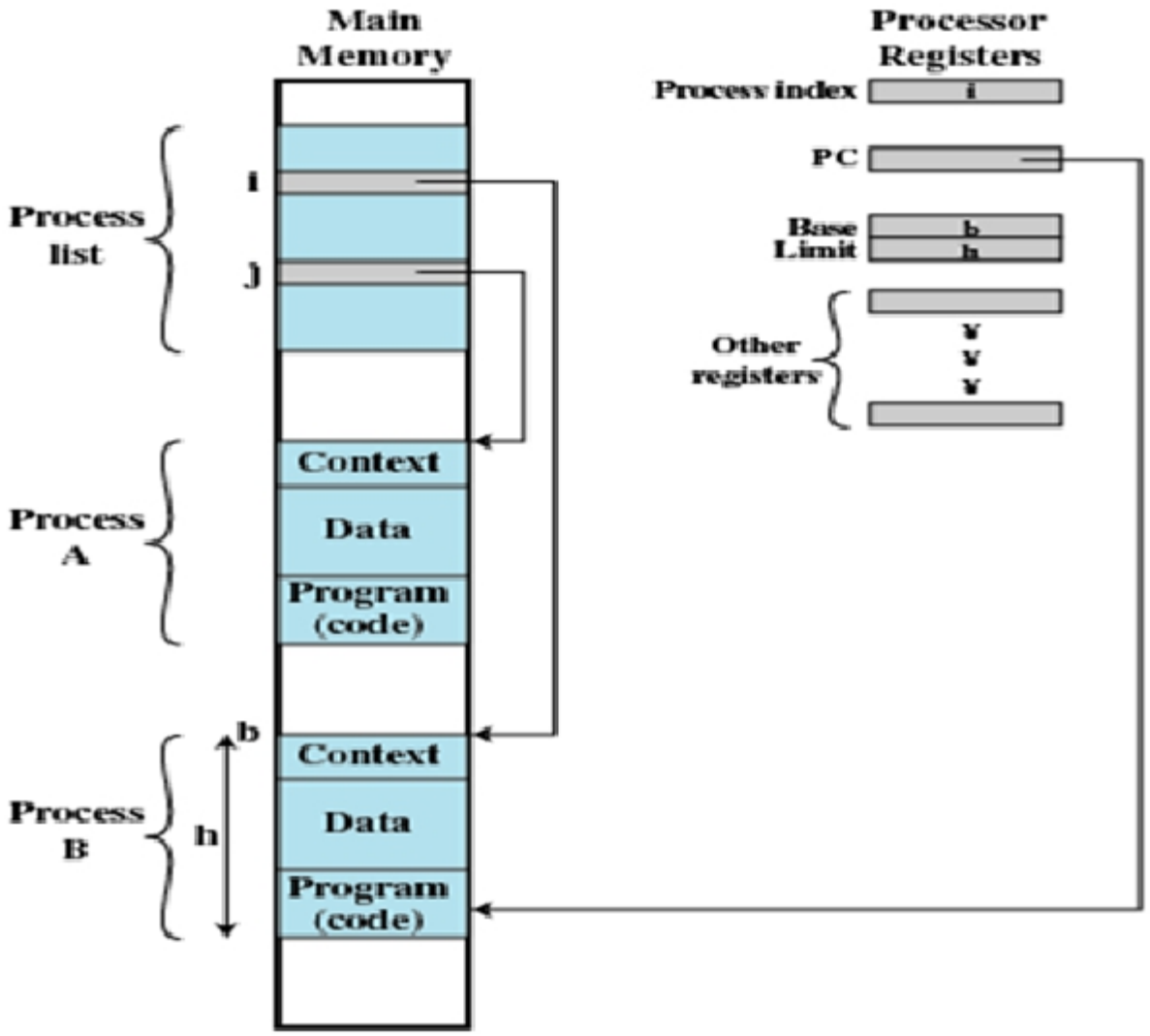
- Stack:
  - Static
  - Last in first out
  - It is used if it is known how much data need to allocate before compile time

Stack VS. Heap



- Heap:
  - Dynamic
  - First in first out
  - It is used if it is not known how much data you will need at runtime.

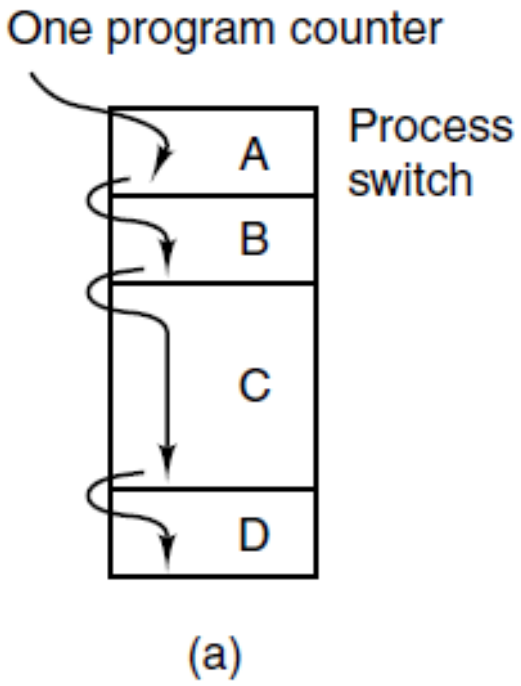
Diagram of Multiple Processes



A process's view of the world

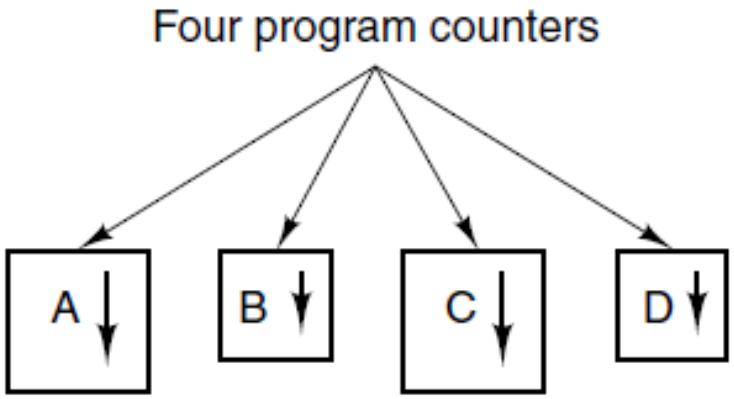
- Multiple processes may run at the same time.
- Each process owns its own *image* of the program.
  - Address space (memory) the program can use
  - State (registers, including program counter & stack pointer)
- Address space and memory protection
  - Each process has its own exclusive address space
    - *(char *) 0xc000 different in P1 & P2
  - Physical memory is divided into user memory and kernel memory; kernel memory can only be accessed when in the kernel mode

The Process Model (1)



(a) Multiprogramming of four programs.

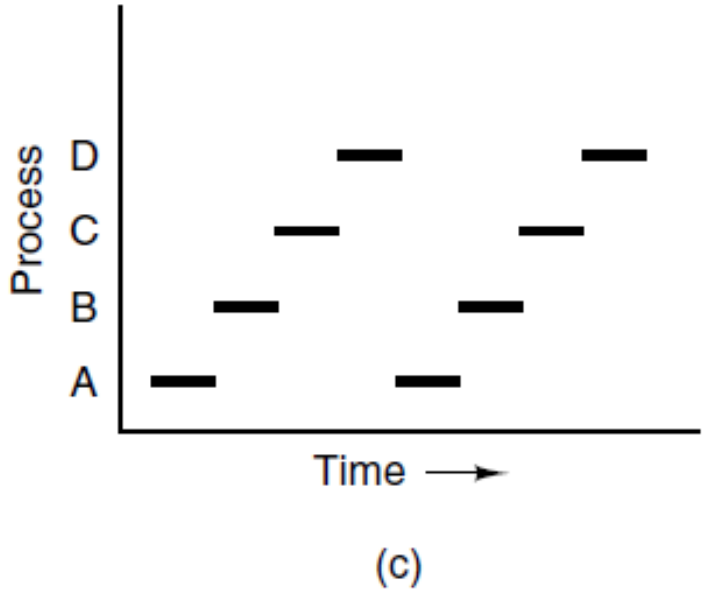
The Process Model (2)



(b)

(b) Conceptual model of
four independent, sequential processes.

The Process Model (3)



(c) Only one program is active at once.

Goals for Today

- Process
  - Concept
  - Operations on Processes
    - Creation
    - Termination
  - Process States
  - Process Implementation

Reasons for Process Creation

Four reasons:

1) System initialization
2) Running process creates another process
3) User creates a new process
4) Initialization of a batch job

Unix Process Creation

- A new process is created by the *fork()* call
- Child and parent are identical
  - *child* returns a 0
  - *parent* returns non-zero
- Often the child executes an execve()
  - change its memory image and run a new program

Process Creation: Unix Example

```
#include <sys/types.h>
#include <errno.h>
#include <stdio.h>
...
pid_t childpid;
...
childpid=fork();
switch(childpid) {
      case -1: fprintf(stderr,"ERROR: %s\n", sys_errlist[errno]);
               exit(1);
               break;
      case 0:
               /* Child's code goes here */
               break;
      default:
               /* Parent's code goes here */
               break;
      }
```

Both parent and child resume from here

Fork()

- A call to fork( ) leads to the execution of two processes, parent and child.
- The child process is an ***almost*** exact copy of the parent process.
- Both processes continue executing from the point where the fork( ) calls returns execution to the main program.

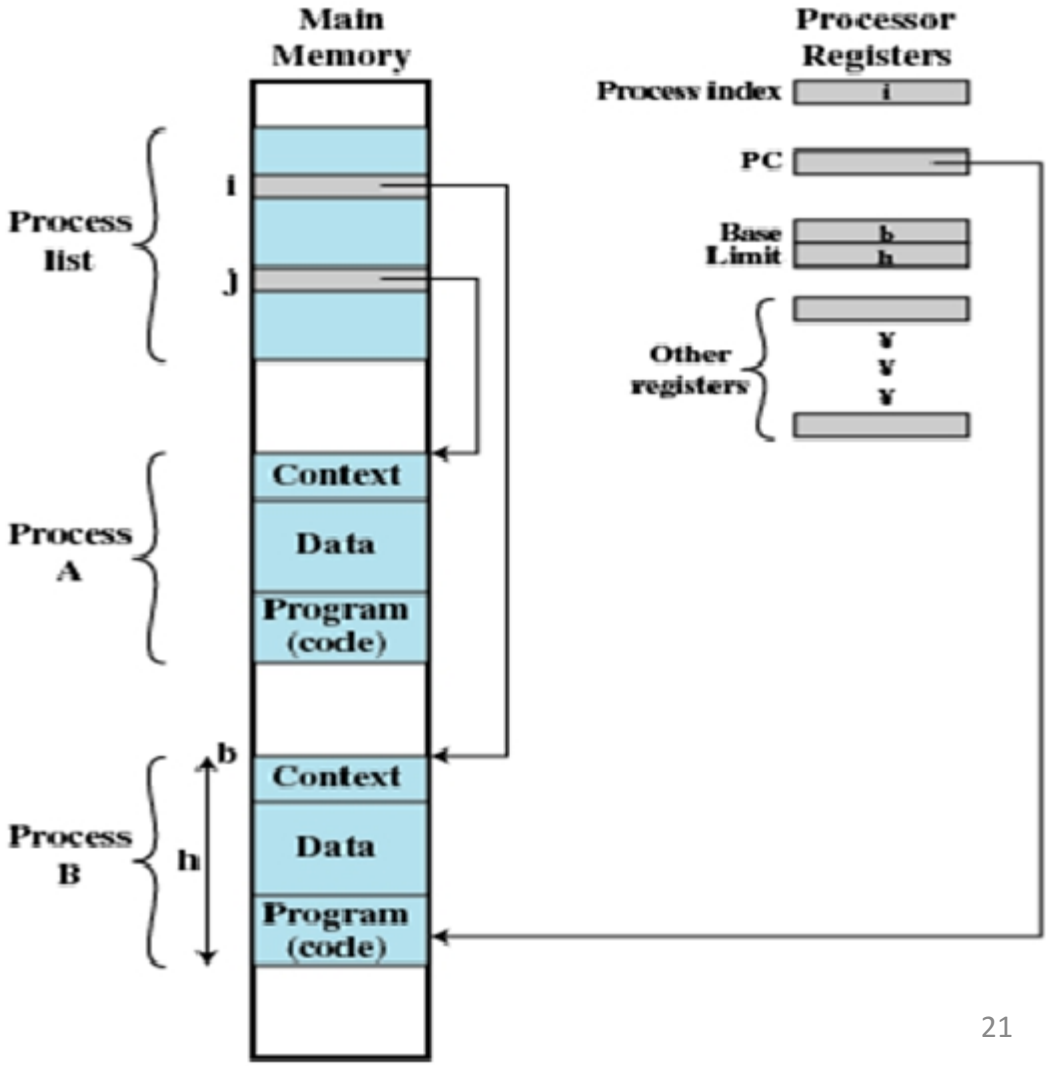
```
main () {
    output A;
    fork()
    Output B;
}
```

```
Parent:
A
B
```

```
Child:
B
```

Differences between the child and parent process

- Different pids
  - Parent: fork( ) returns the pid of the child process
  - Child: fork( ) returns 0
- Separate copies of data
- Separate program counter (PC) indicates where to execute next



Global Variables

- After fork(), the global variable in both parent and child processes have the same value.
- If one changes it, the other won't see it changed.

Code Fun

```
#include <iostream>
#include <unistd.h>

int main()
{
    int x=1, y=2;

    x = fork();
    y = fork();

    if(x!=0) printf("Type 1\n");
    if(y!=0) printf("Type 2\n");

    if(x>0 || y>0) printf("Type 3\n");
    if(x==0 && y==0) printf("Type 4\n");
    if(x!=0 && y!=0) printf("Type 5\n");

    return 0;
}
```

What is the output of this function?