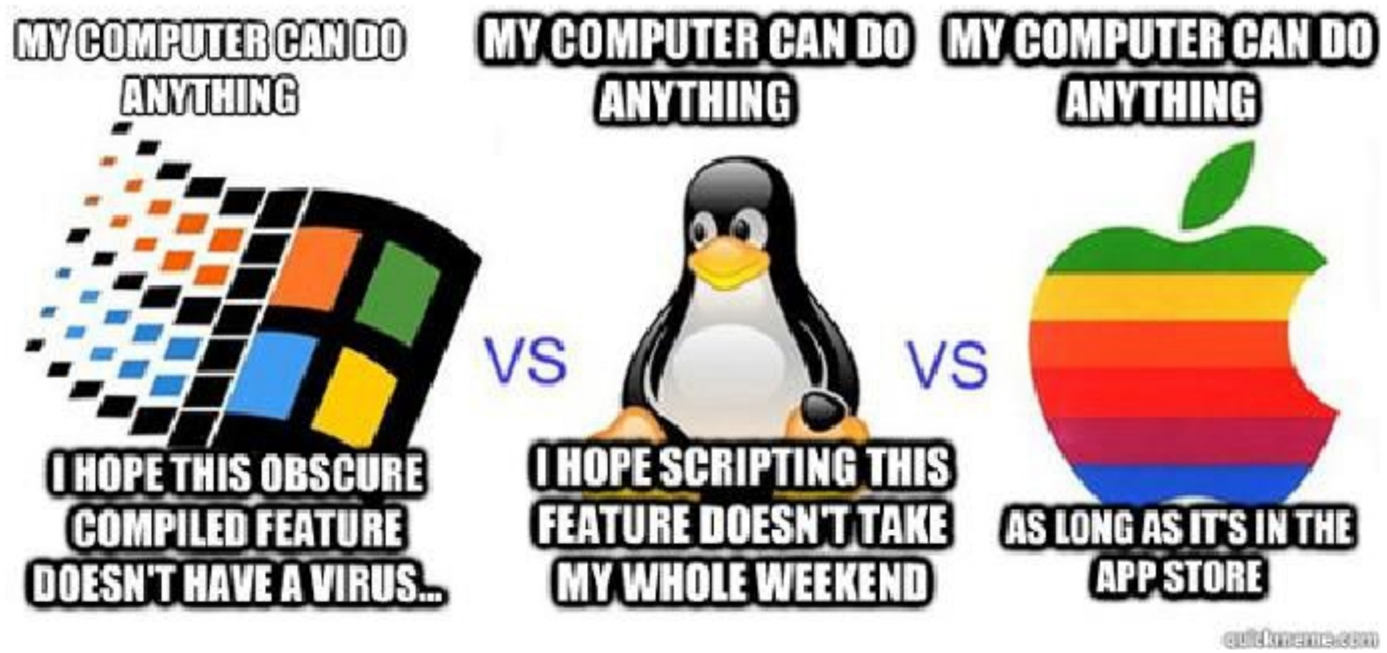
CS 492: Operating Systems

Processes (2)

Instructor: Iraklis Tsekourakis

Email: itsekour@stevens.edu

Processes (Part II)


MY COMPUTER CAN DO ANYTHING
I HOPE THIS OBSCURE COMPILED FEATURE DOESN'T HAVE A VIRUS...
VS
MY COMPUTER CAN DO ANYTHING
I HOPE SCRIPTING THIS FEATURE DOESN'T TAKE MY WHOLE WEEKEND
VS
MY COMPUTER CAN DO ANYTHING
AS LONG AS IT'S IN THE APP STORE
quickmeme.com

Last Lecture

- Process
  - Concept
  - Operations on Processes
    - Creation
    - Termination
  - Process States
  - Process Data Structure

Fork()

- Fork( ) results in the execution of two processes, parent and child.
- The child process is an almost exact copy of the parent process.

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

```
Parent:
A
B
```

```
Child:
B
```

Anything Wrong with this Function?

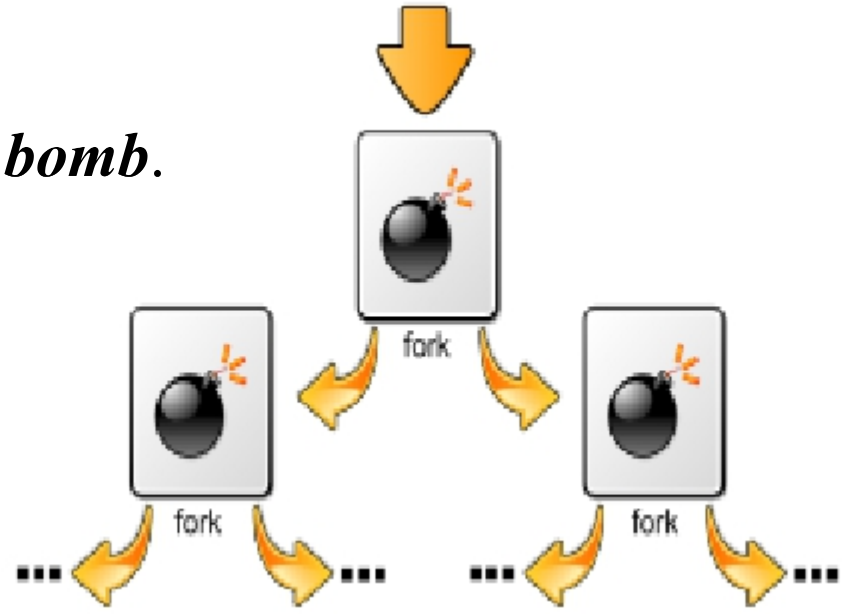
```
#include <unistd.h>
#include <iostream>
using namespace std;

int main()
{
    while(1)
            fork();
    return 0;
}
```

Possible Chaos: Fork Bombs

- A process that creates a child process, and then both of these processes will create child processes, and then all of these processes will create child processes, and so on until the system is overwhelmed.

- Such a program is called *a **fork bomb***.



Fork Bomb: Example 1

```
#include <unistd.h>
#include <iostream>
using namespace std;

int main()
{
    while(1)
          fork();
    return 0;
}
```

(Semi-)Fork Bomb: Example 2

```
#include <unistd.h>
#include <iostream>
using namespace std;

int main()
{
        cout << "0. I am process " << getpid() << endl;
        (void) fork();
        cout << "1. I am process " << getpid() << endl;
}
```

What will be the ouput?

Given n fork() calls in the code,
(1) how many processes are executed?
(2) how many output lines there will be?

```
int main()
{
        cout << "I am process " << getpid() << endl;
        (void) fork();
        cout << "I am process " << getpid() << endl;
        (void) fork();
        cout << “I am process " << getpid() << endl;
         ….
        cout << “I am process”  << getpid() << endl;
}
```

Prevention of Fork Bombs

- One way of preventing a fork: limit the max number of processes that a single user may own.
  - Use the *ulimit* command: e.g. ulimit -Hu 30

Process Creation in Windows OS System

- Process created by *CreateProcess* call
- *CreateProcess* loads a program that is specified in the call
- Child and parent processes are NOT necessarily the same
- Child process executes concurrently with parent

Windows CreateProcess()

C++

```cpp
BOOL WINAPI CreateProcess(
  _In_opt_     LPCTSTR               lpApplicationName,
  _Inout_opt_  LPTSTR                lpCommandLine,
  _In_opt_     LPSECURITY_ATTRIBUTES lpProcessAttributes,
  _In_opt_     LPSECURITY_ATTRIBUTES lpThreadAttributes,
  _In_         BOOL                  bInheritHandles,
  _In_         DWORD                 dwCreationFlags,
  _In_opt_     LPVOID                lpEnvironment,
  _In_opt_     LPCTSTR               lpCurrentDirectory,
  _In_         LPSTARTUPINFO         lpStartupInfo,
  _Out_        LPPROCESS_INFORMATION lpProcessInformation
);
```

Windows CreateProcess() Example

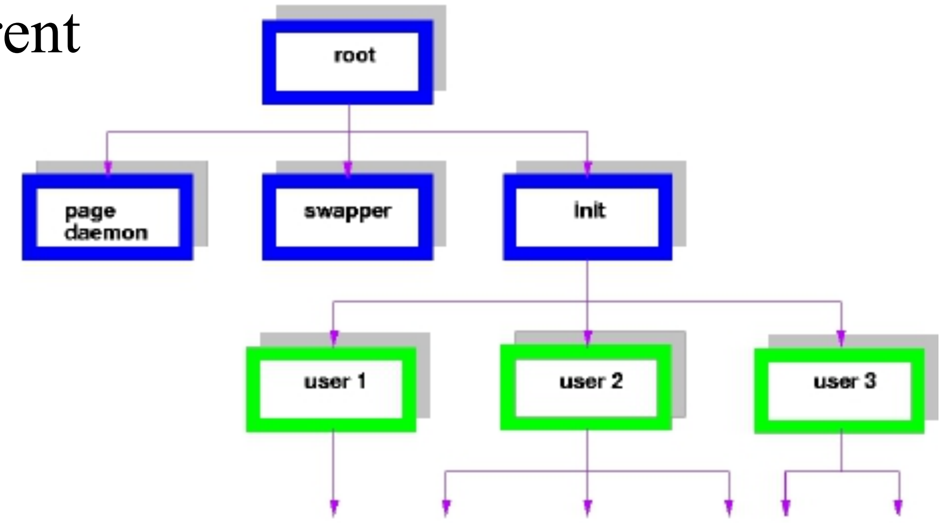
```
PROCESS_INFORMATION ProcessInfo; //This is what we get as an [out]
parameter
STARTUPINFO StartupInfo; //This is an [in] parameter
ZeroMemory(&StartupInfo, sizeof(StartupInfo));
StartupInfo.cb = sizeof StartupInfo ; //Only compulsory field

if(CreateProcess("c:\\winnt\\notepad.exe", NULL,
    NULL,NULL,FALSE,0,NULL,
    NULL,&StartupInfo,&ProcessInfo))
{
    WaitForSingleObject(ProcessInfo.hProcess,INFINITE);
    CloseHandle(ProcessInfo.hThread);
    CloseHandle(ProcessInfo.hProcess);
}
else
{
    MessageBox("The process could not be started...");
}
```

Example: Create a new Notepad process by using CreateProcess()

Process Hierarchies

- Unix:
  - Parent and children processes form a hierarchy
  - If a process exits, its children are “inherited” by the exiting process’s parent



- Windows: no concept of process hierarchy
  - All processes are created equal

Process Termination

- Reasons for termination
  1) Normal exit
  2) Error exit (e.g., “file doesn’t exist” error)
  3) Fatal error (e.g., program bug)
  4) Killed by another process
- How to terminate?
  - Unix: system call *kill()*
  - Windows: system call *TerminateProcess()*
  - For both Unix and Windows, termination of parent processes does not kill their children processes

Goals for Today

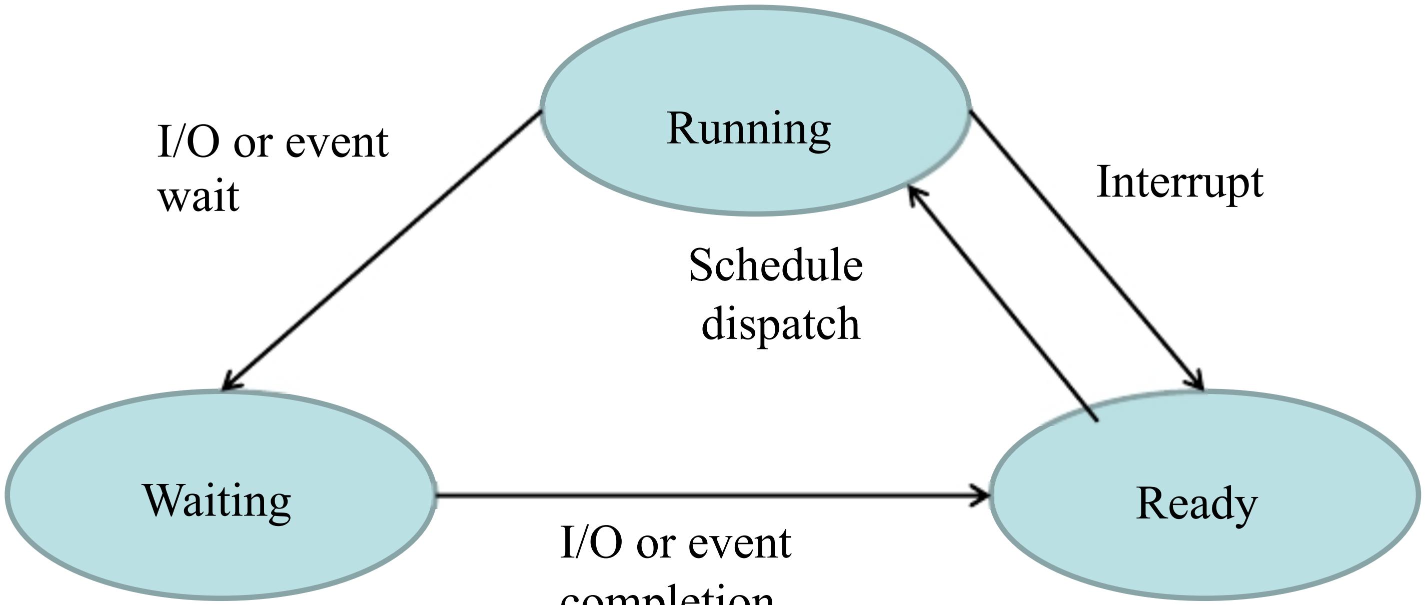
- Process
  - Concept
  - Operations on Processes
  - Process States
  - Process Data Structure

Process State

- A process has an execution state that indicates what it is currently doing
  - Running: Executing instructions on the CPU
    - It is the process that has control of the CPU
    - Question: for 1-CPU system, how many processes can be in the running state simultaneously?
  - Ready: Waiting to be assigned to the CPU
    - Ready to execute, but another process is executing on the CPU
  - Waiting (also called as blocked): Waiting for an event, e.g., I/O completion
    - It cannot make progress until event is signaled (disk completes)

Process State Transition

- Four transitions

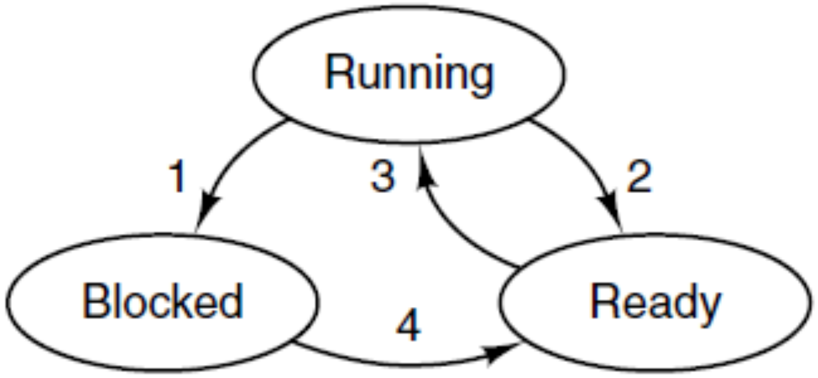


How to Check Process States in Unix/Linux?

- As a process executes, it moves from state to state
  - Unix “ps”: STAT column indicates execution state
  - What state do you think a process is in most of the time?

Question?

- In theory, with three states, there could be 6 transitions. However only 4 transitions are shown. Are there any circumstances in which either or both of the missing transitions might occur?



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Goals for Today

- Process
  - Concept
  - Operations on Processes
  - Process States
  - Process Data Structure

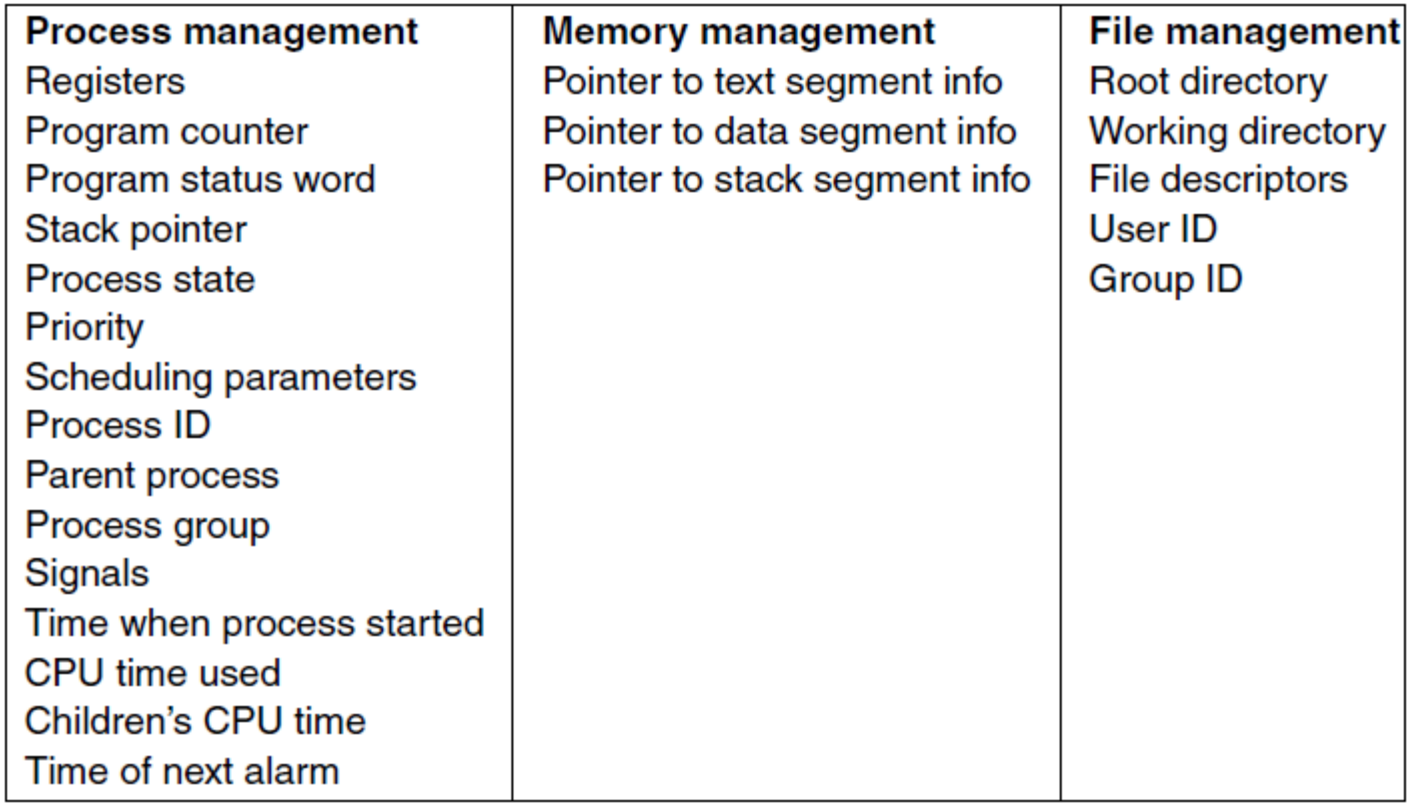
Process Data Structures

- How does the OS represent a process in the kernel?
  - Multiple processes exist in the system, each in its particular state
  - The OS data structure representing each process is called the Process Control Block (PCB)
  - The Process Table, has one entry per process, the PCB
  - The PCB contains all of the info about a process

What is in PCB?

- Each process consists of
  - PID
  - Register values for the process (including PC and SP values for the process)
  - The address space of the process
  - Process priority
  - Pointer to the next PCB
  - I/O Information

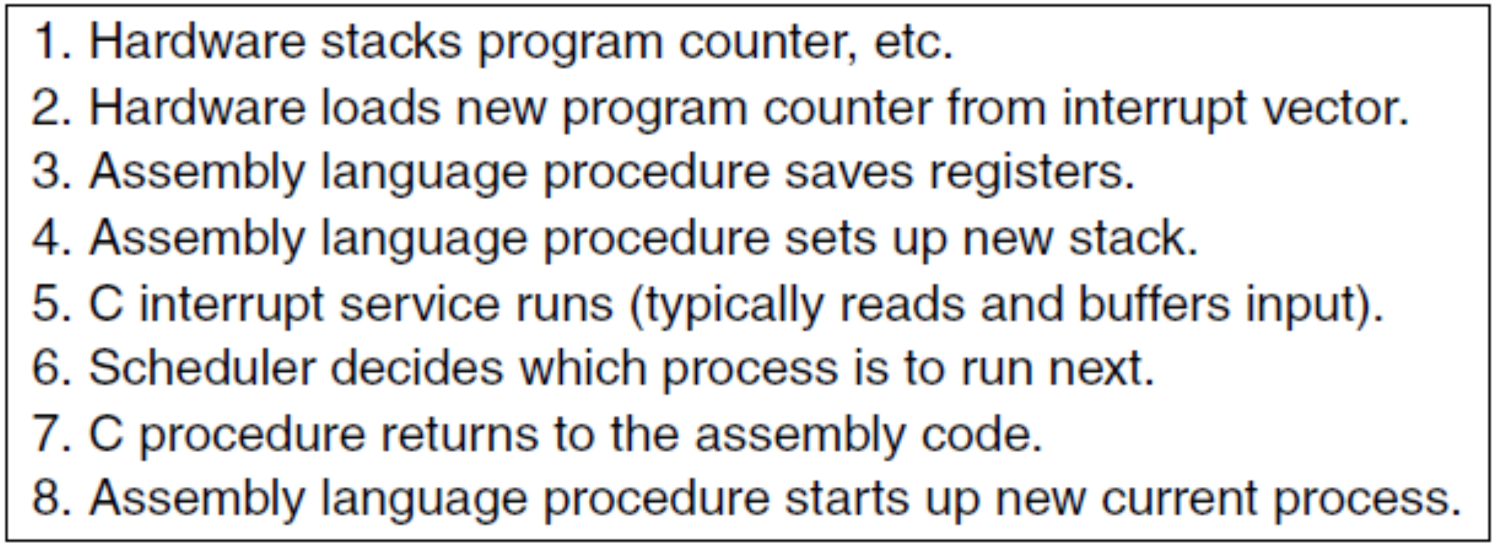
Processes Implementation



Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	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		

Some of the fields of a typical process table entry.

Process Implementation (2)

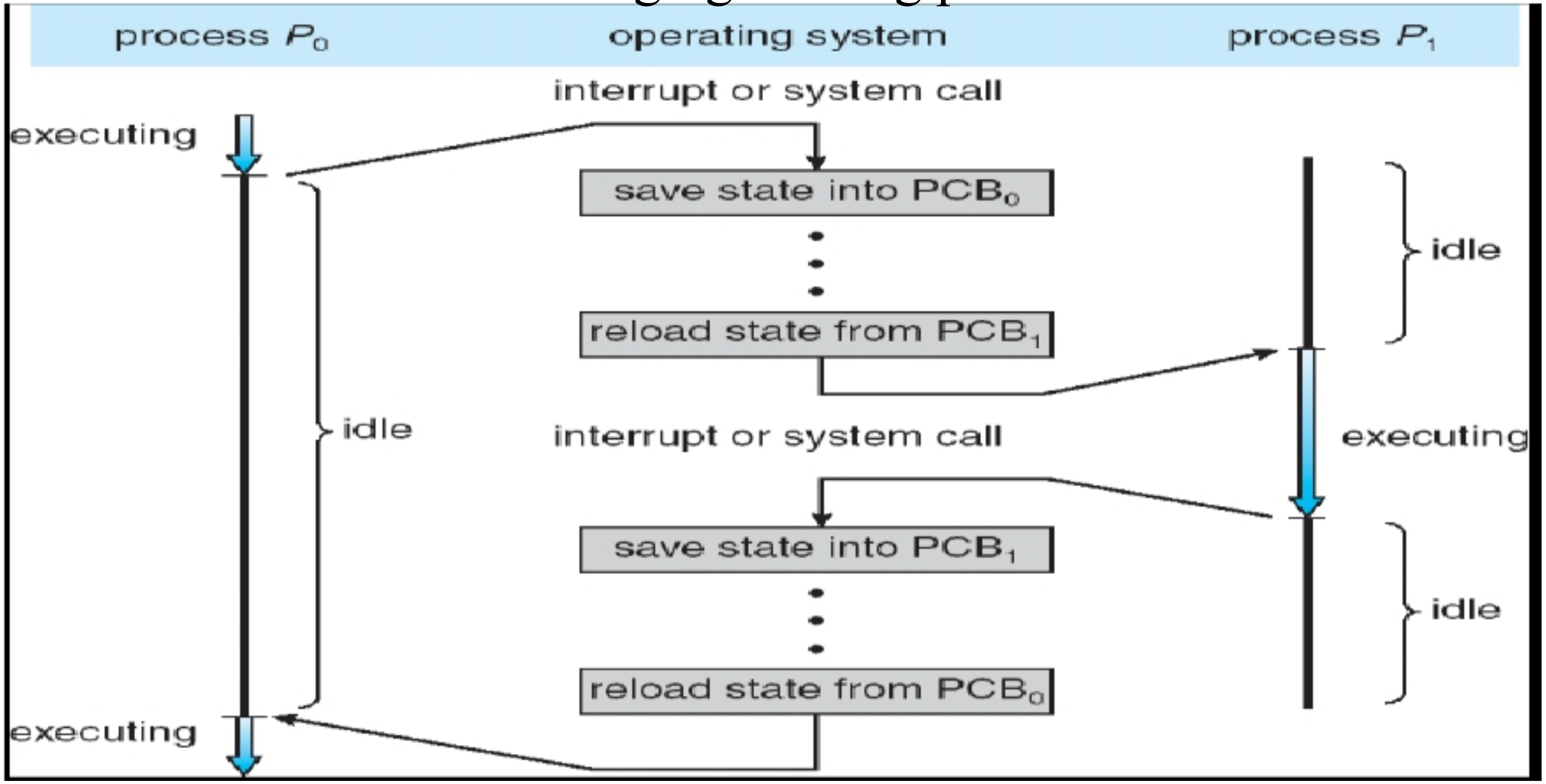


1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly language procedure saves registers.
4. Assembly language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly language procedure starts up new current process.

Skeleton of what the lowest level of the operating system does when an interrupt occurs.

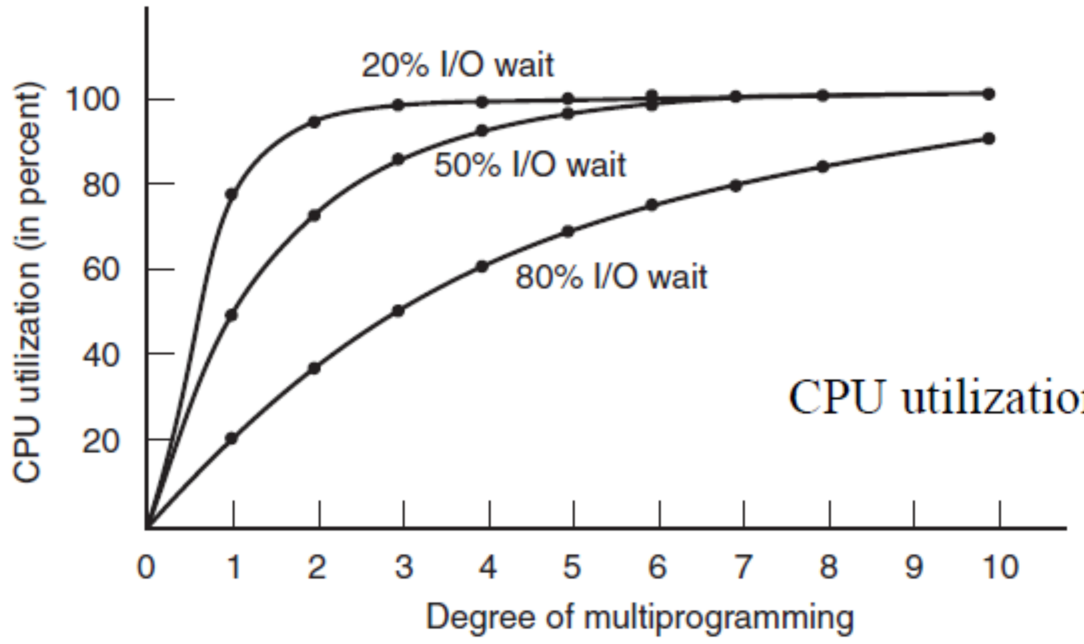
Context Switch

- *Context switch: c*hanging running proc



Q: when the context is switched from P0 to P1, what's the state of P0?

Modeling Multiprogramming



$$\text{CPU utilization} = 1 - p^n$$

CPU utilization as a function of the number of processes in memory.

Modeling Multiprogramming (2)

- Model may be simple-minded, but although approximate, can be used for CPU performance prediction
  - 512MB RAM, 128MB for OS, 128MB for each user-program, 80% wait–> CPU util:
  - 1024MB RAM -> CPU util:
  - 1568MB RAM -> CPU util: