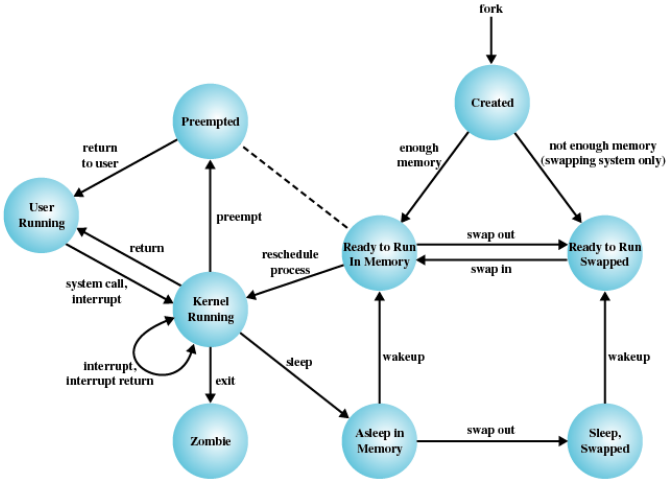
Operating Systems & Computer Networks
Processes



Content (1)

Definitions of a Process

- Program in execution

- Instance of a program running on a computer
  - There may be multiple instances of the same program, each as a separate process

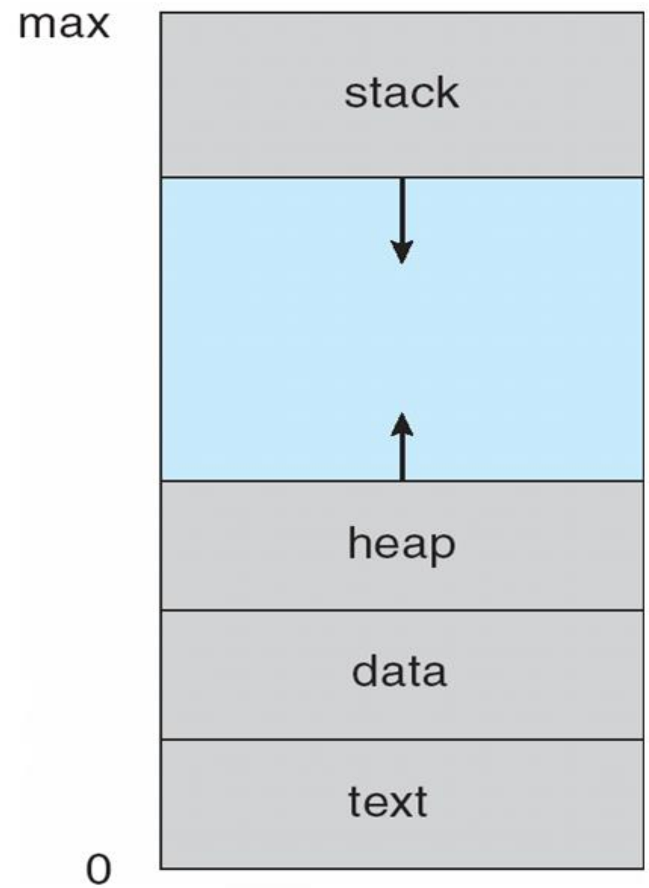
- Unit characterized by
  - Execution of a sequence of instructions
  - Current state
  - Associated block of memory

Related Concepts to “Process”

- Thread: One (of several) runtime entities that share the same address space
  - Easy cooperation, requires explicit synchronization
  - A process may consist of several threads
- Application: User-visible entity, one or more processes

Program vs. Process

- Multiple parts
  - Program code → text section
  - Current activity → program counter, processor registers
  - Stack → temporary data
  - Data section → global variables
  - Heap → dynamic memory
- Program is passive entity, process is active
  - Program becomes process when executable file loaded into memory
- One program can be several processes



Tasks of an OS concerning processes

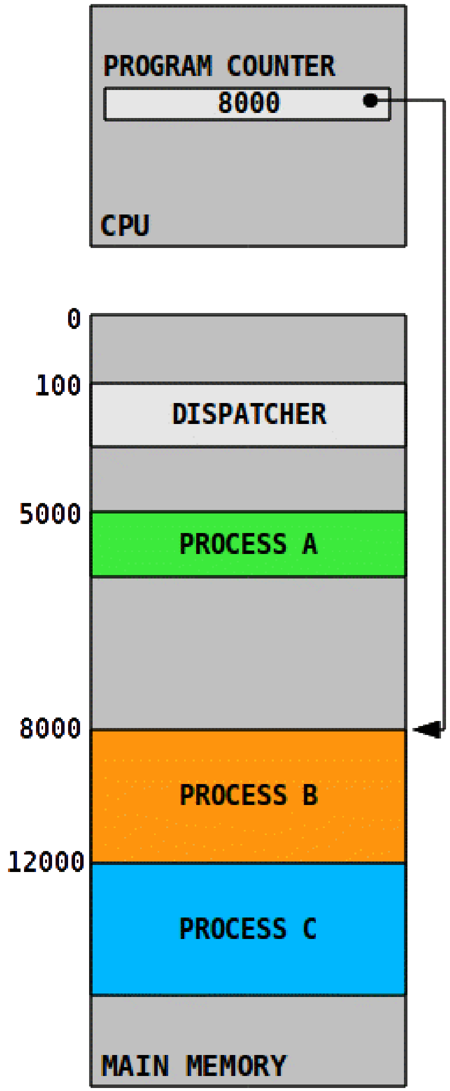
- Interleaved execution (by scheduling) of multiple processes
  - Maximization of processor utilization
  - Reduction of response time

- Allocation of resources for processes
  - Consideration of priorities
  - Avoidance of deadlocks

- Support for Inter-Process Communication (IPC)

- On-demand user-level process creation
  - Structuring of applications

Process execution (Trace)

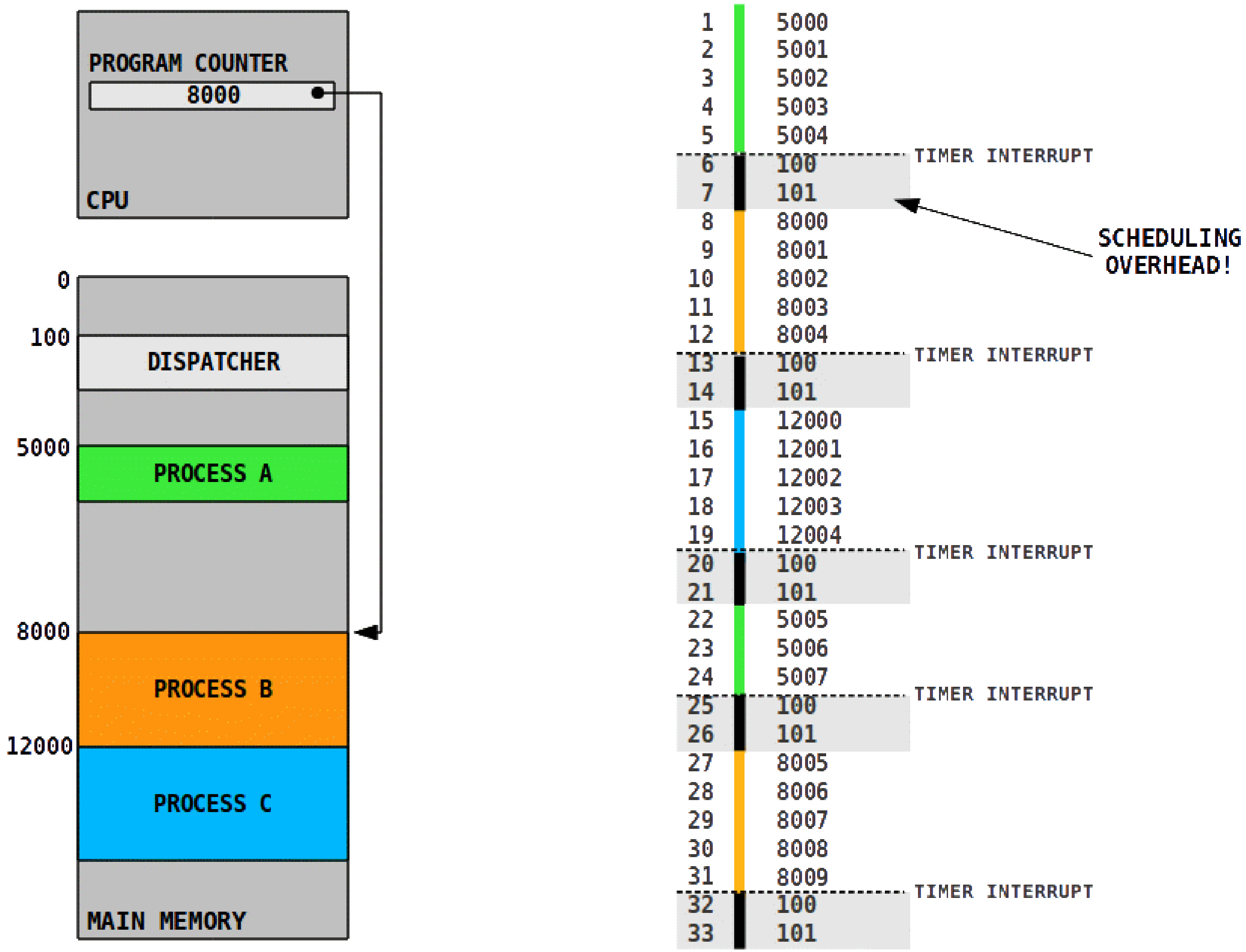


5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011
(a) Trace of Process A	**(b) Trace of Process B**	**(c) Trace of Process C**

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

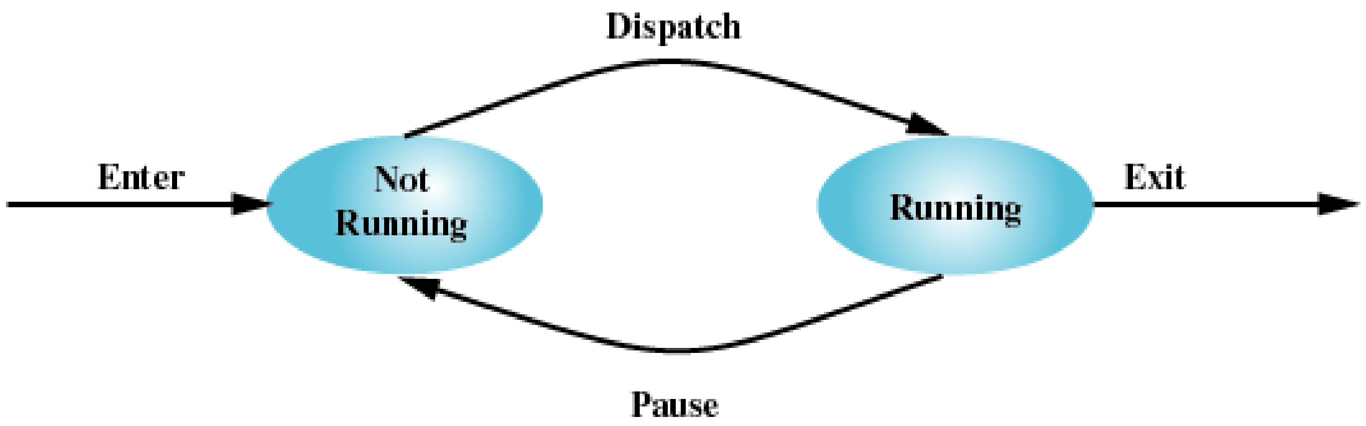
Figure 3.3 Traces of Processes of Figure 3.2

Process execution (Trace)



Simple Process Model

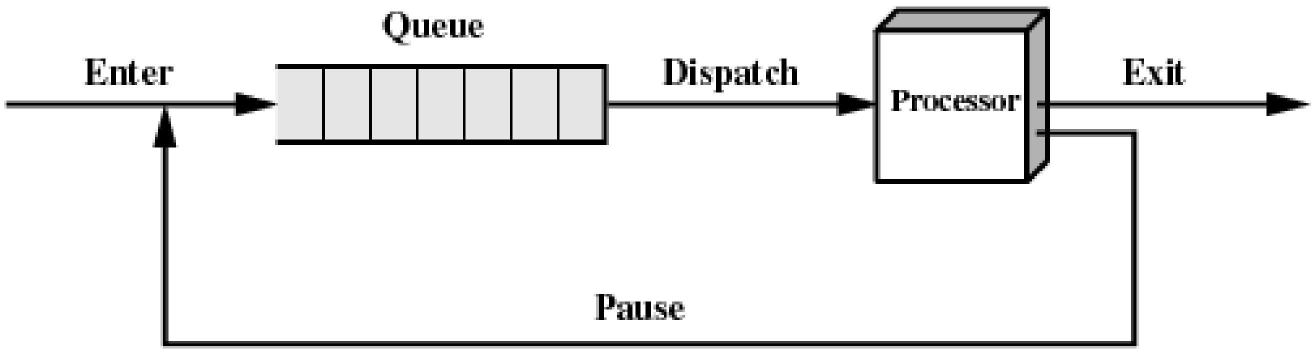
- Process is in one of two states:
  - running
  - not running



- How to implement?

Simple Process Model

- Running processes managed in queue:



- What information required?

Process Control Block (PCB)

Definition: OS data structure which contains the information needed to manage a process (one PCB per process)

Process identifiers	• IDs of process, parent process, and user

CPU state	• User-visible registers
	• Control and status registers:
	  • Stack pointer (SP)
	  • Program counter (PC)
	  • Processor status word (PSW)

Control information	• Scheduling information:
	  • Process state, priority, awaited event
	• Accounting information:
	  • Amount of memory used, CPU time elapsed
	• Memory management:
	  • Location and access state of all user data
	• I/O management:
	  • Devices currently opened (files, sockets)

Process Control Block (PCB)

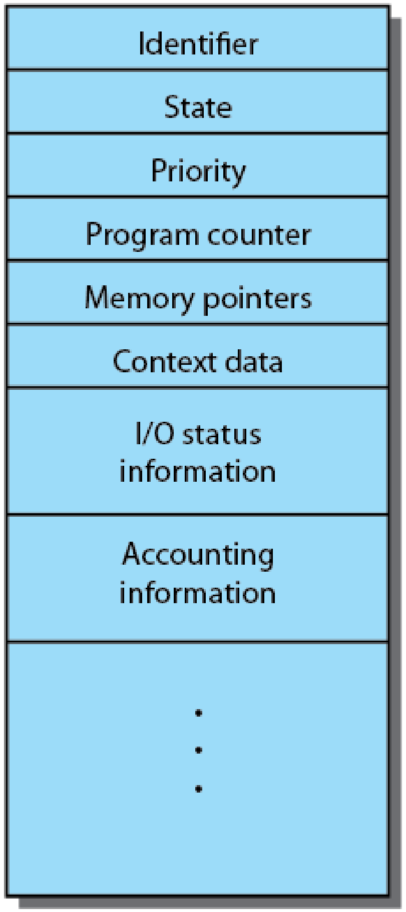


Figure 3.1 Simplified Process Control Block

Reasons for Process Creation

- Interactive logon
  - User logs onto a terminal
  - ➢ May create several processes as part of logon procedure (e.g. GUI)
- Created by the OS to provide a service
  - Provide a service to user program in the background (e.g. printer spooling)
  - ➢ Either at boot time or dynamically in response to requests (e.g. HTTP)
- Spawned at application start-up
  - Separation of a program into separate processes for algorithmic purposes

➢ Always spawned by existing process
  - Operating system creates first process at boot time
  - Processes are organized in a tree-like structure (`` `pstree` ``)

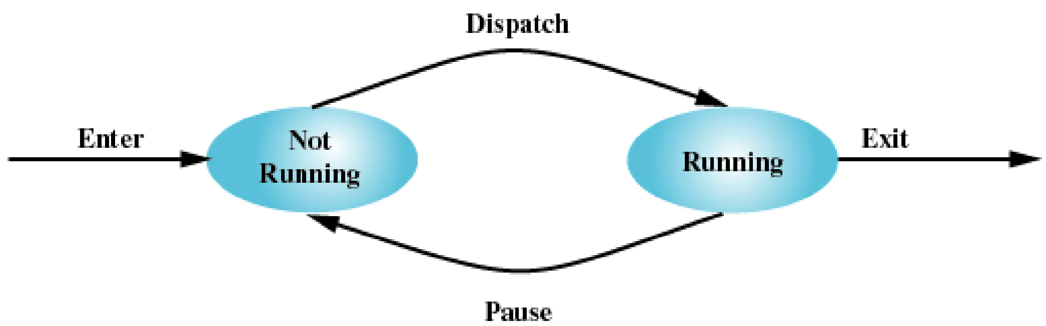
Process Termination

- Execution of process is completed
  - process terminates itself by system call
- Other user process terminates the process
  - Parent process or other authorized processes
- OS terminates process for protection reasons
  - Invalid instruction (process tries to execute data)
  - Privileged instruction in user mode
  - Process tries to access memory without permission
  - I/O-Error
  - Arithmetic error

➢ Some exceptions can be caught and handled by the process.

Process Model

- Simple model with two states



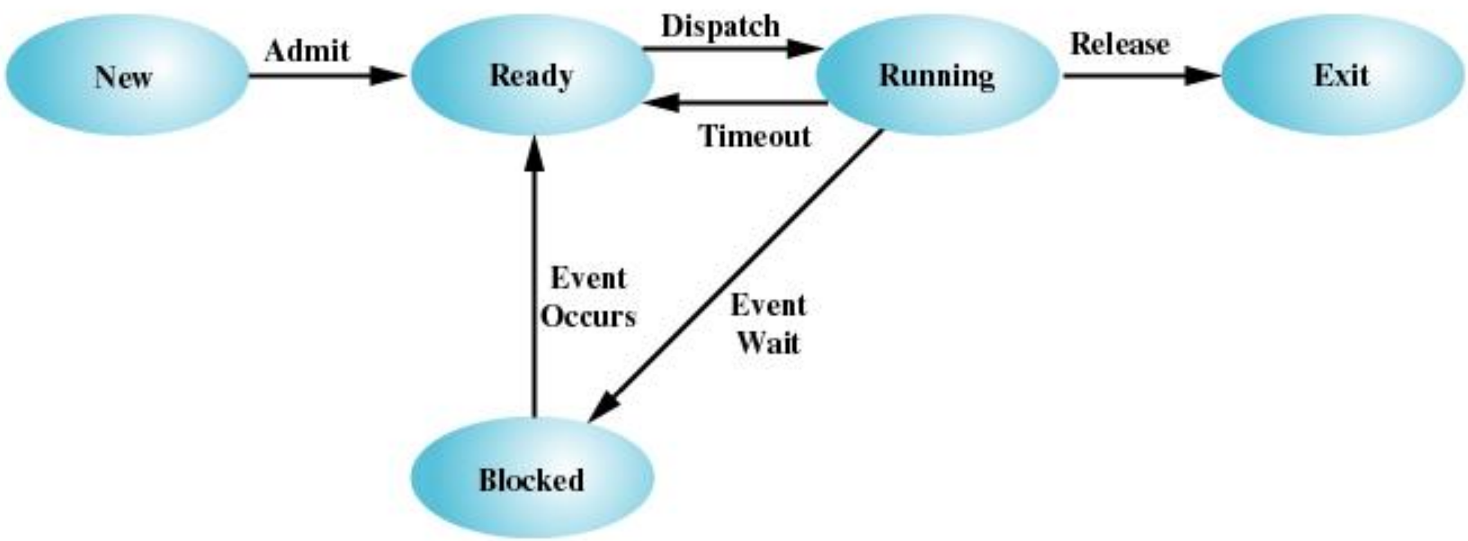
- Problems
  - Most of the processes will be waiting for IO
  - Different IO devices
  - Different priorities

➢ Extend the model

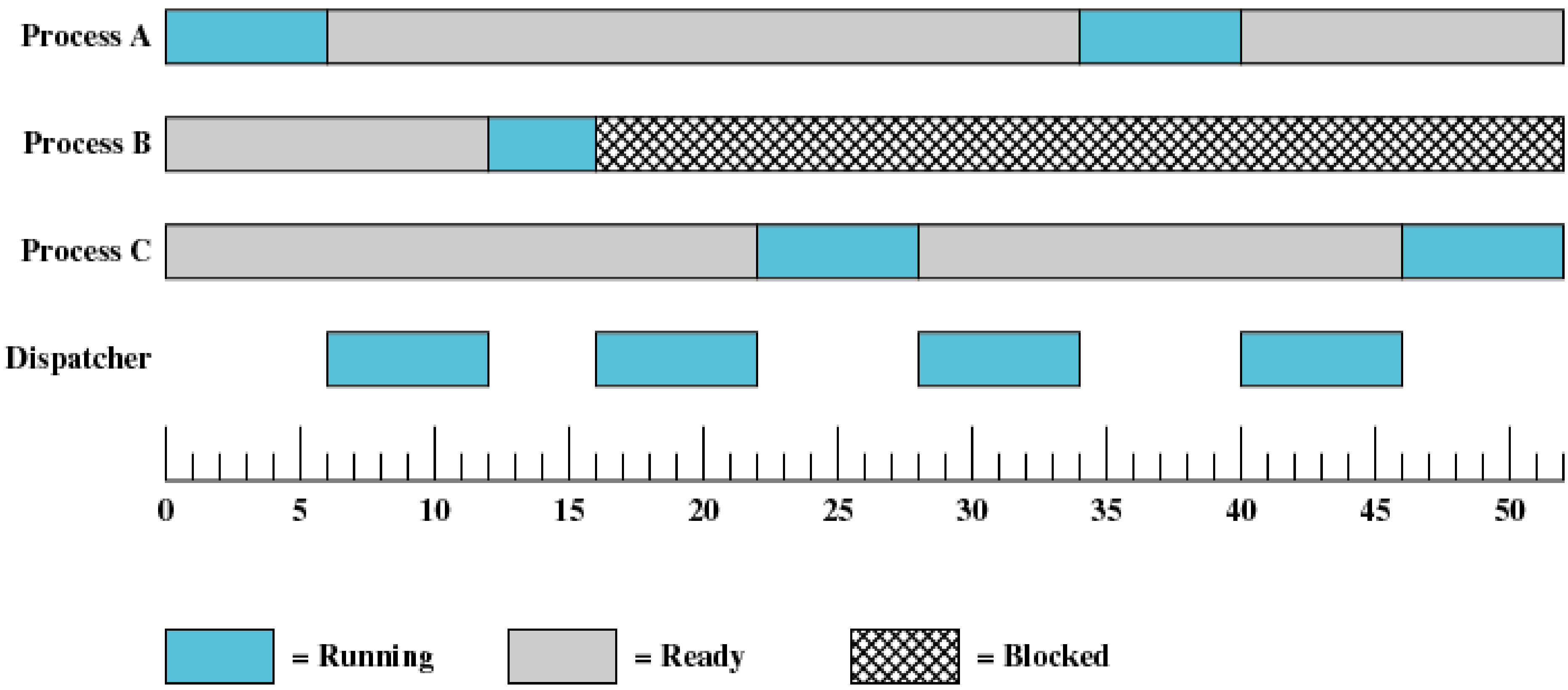
Extended Process Model

Five states including creation, termination, and resource handling:

- **Running:** currently being executed
- **Ready:** ready to run, waiting for execution
- **Blocked:** not ready to run, waiting for external event, e.g., completion of I/O operation
- **New:** newly created process, not yet in running set
- **Exit:** completed/terminated process, removed from running set

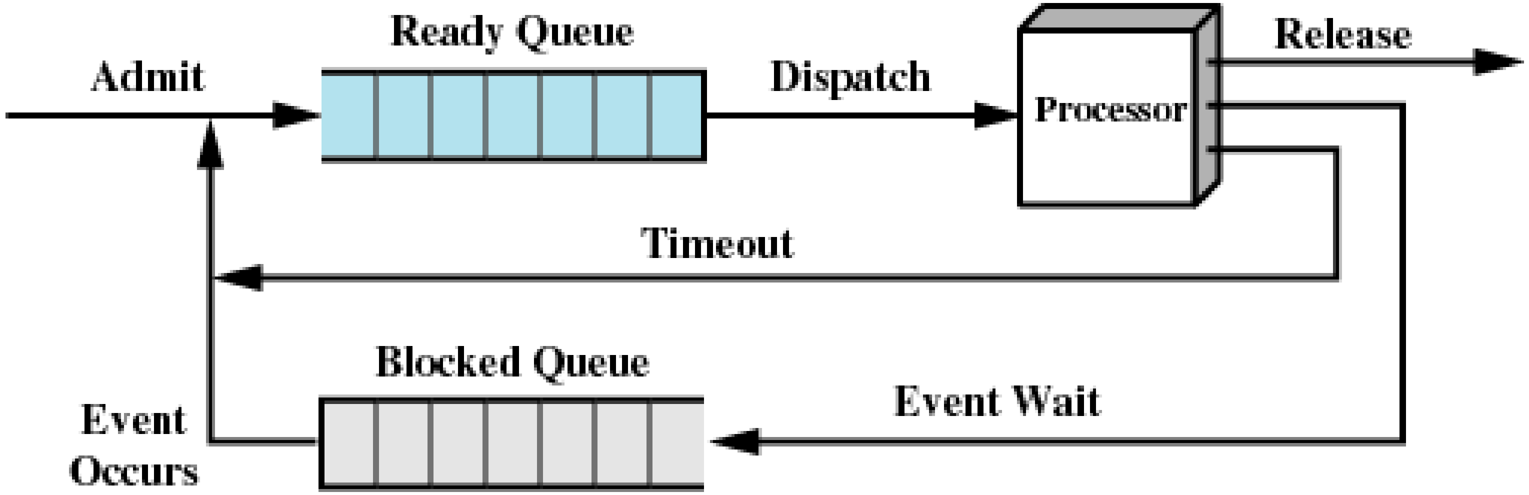


Process States over Time



Implementation of Process States

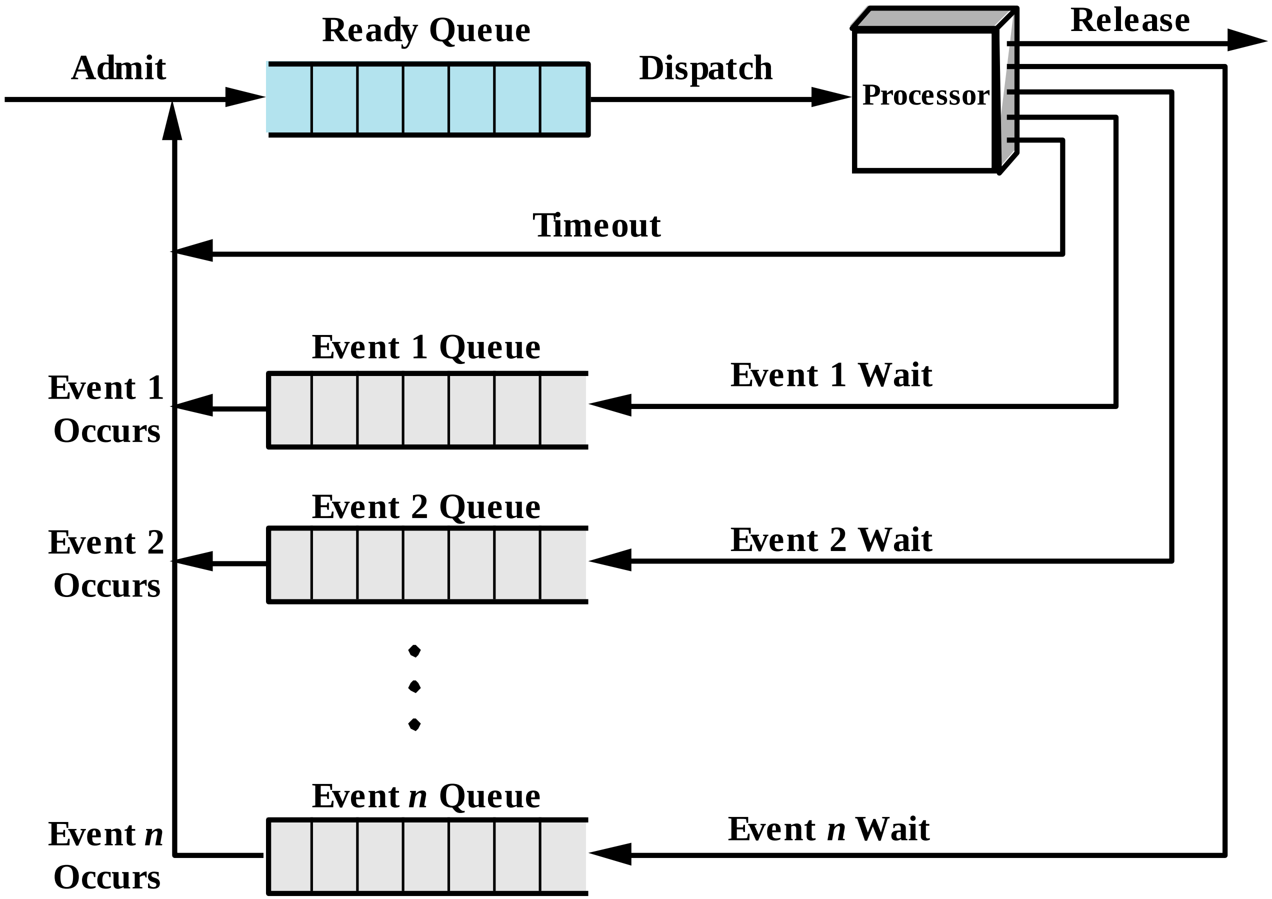
- Assign process to different queues based on state of required resources
- Two queues:
  - Ready processes (all resources available)
  - Blocked processes (at least one resource busy)



➤ But what happens if processes need *different* resources?

Improved Implementation

- Several queues one for each resource / type of resource



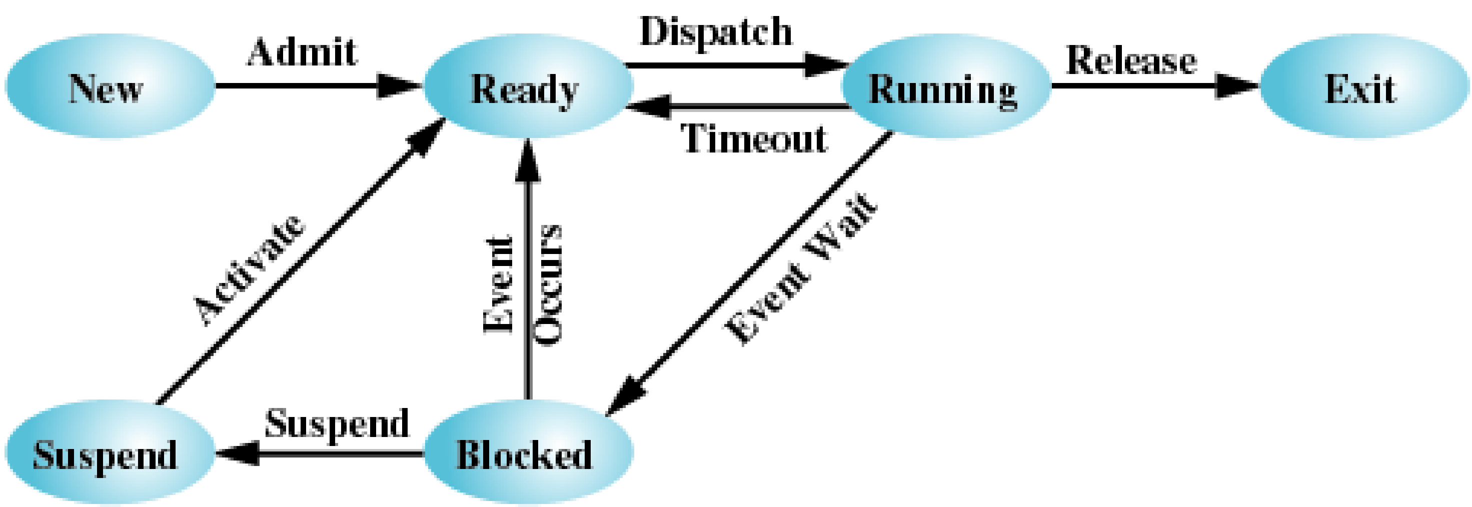
➢ More efficient, but fairness issues must be considered

Suspension / Swapping of Processes

Swapping motivated by two observations:

- Physical main memory is (was) a scarce resource
- Blocked processes may wait for longer periods of time (e.g. during I/O, while waiting for requests, ...)

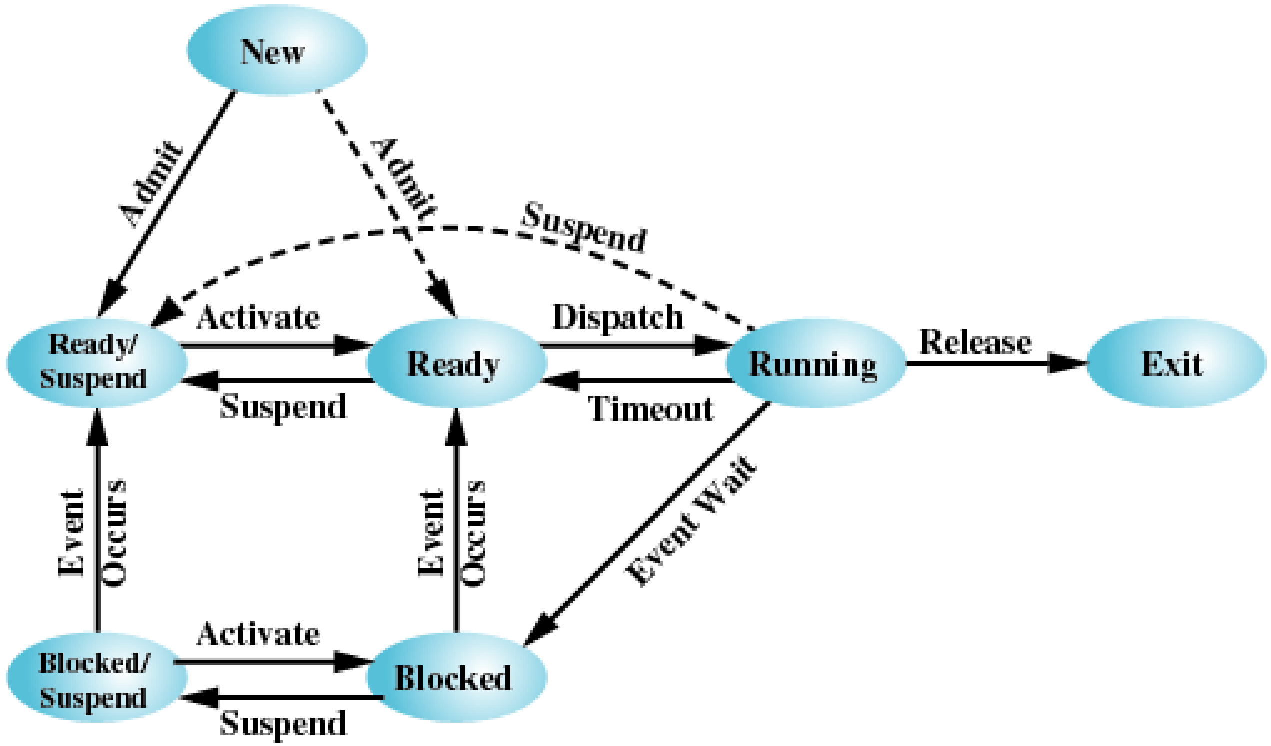
➢ Swap blocked processes to secondary storage thereby reducing memory usage



Extended Process State Diagram

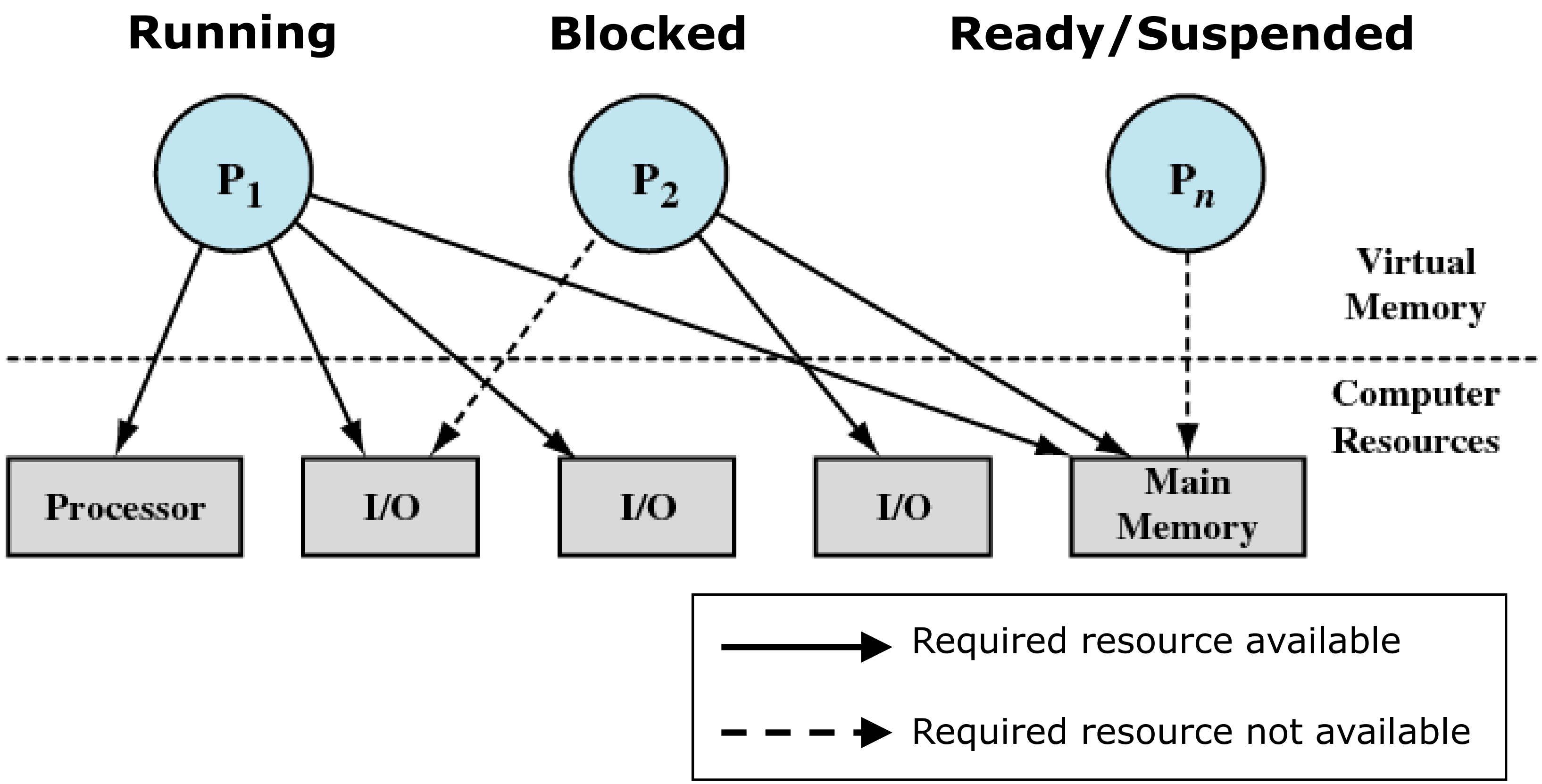
Two additional considerations:

- Blocked/swapped processes may become ready to run when event occurs
- Ready and/or running processes may be swapped even without waiting for event



Processes and Resource Allocation

- Process state reflects allocated resources:

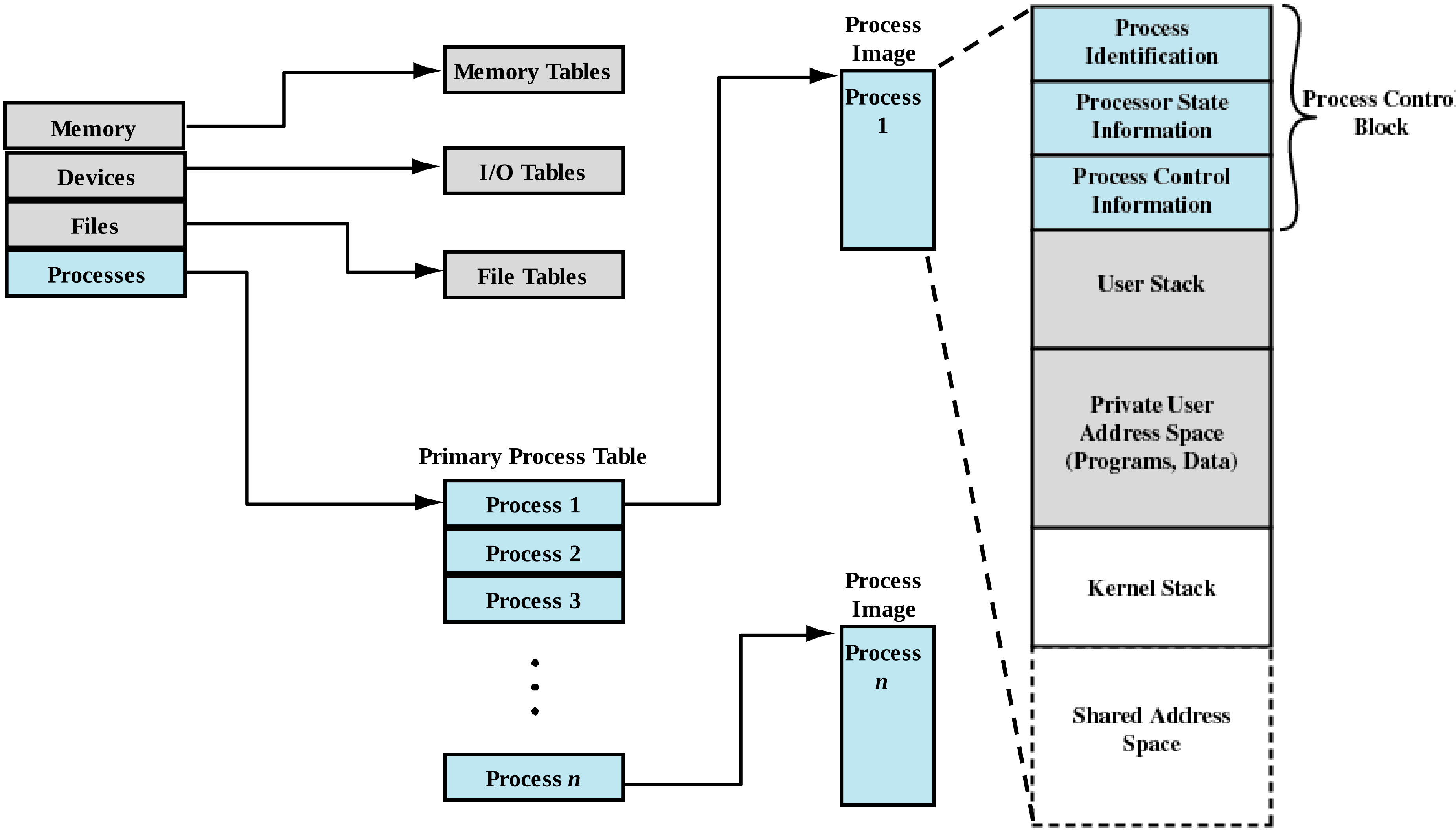


Data Structures

Global data structures for processes and resources usage:

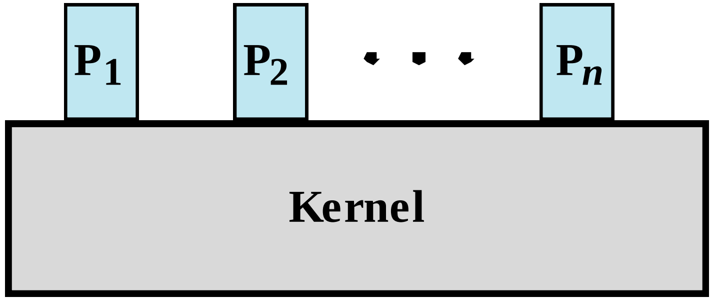
- Process tables:
  - Process Control Block (PCB)
  - Location of process image in memory
  - Resources (process-specific view)
- Memory tables:
  - Allocation of primary and secondary memory
  - Protection attributes of blocks of (shared) memory
  - Virtual memory management
- I/O tables:
  - Allocation of I/O devices, assignment to processes
  - State of current operation and corresponding memory region
- File tables:
  - Currently open files
  - Location on storage media / secondary memory
  - State and attributes

Process Control Table and Image



Kernel / Process Implementations

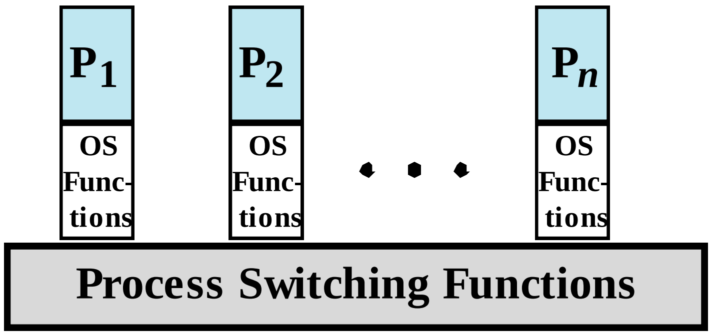
- Separated kernel and processes:
  - Separate memory and stack for kernel
  - Kernel is no process
  - ➢ Expensive and unsafe



(a) Separate kernel

Kernel / Process Implementations

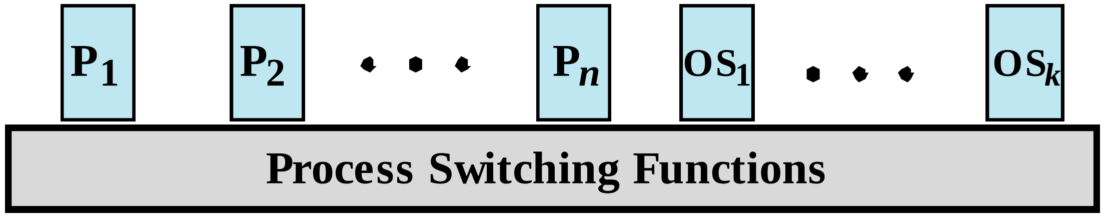
- Execution of system calls as part of user process, but in kernel mode:
  - Kernel functions use same address space
  - Same process switches into privileged mode (Ring 0)
  - ➢ Less expensive and quite safe



(b) OS functions execute within user processes

Kernel / Process Implementations

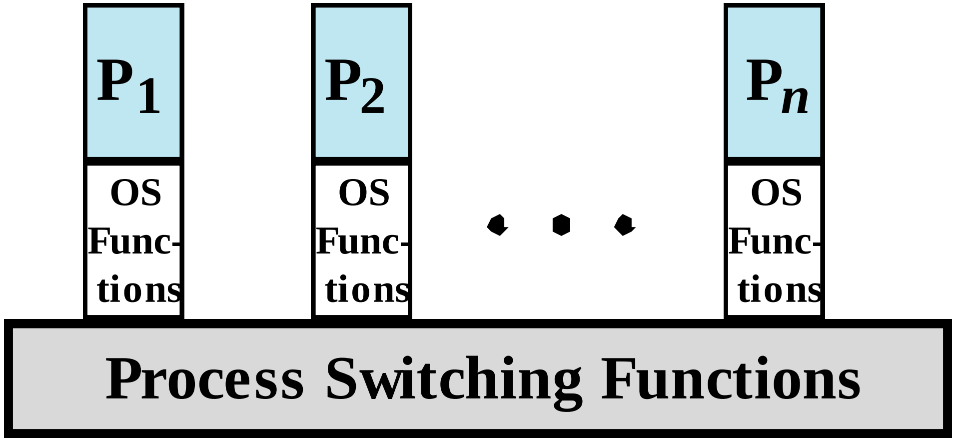
- Microkernel:
  - Collection of system processes that provide OS services
  - ➢ Quite expensive but very safe



(c) OS functions execute as separate processes

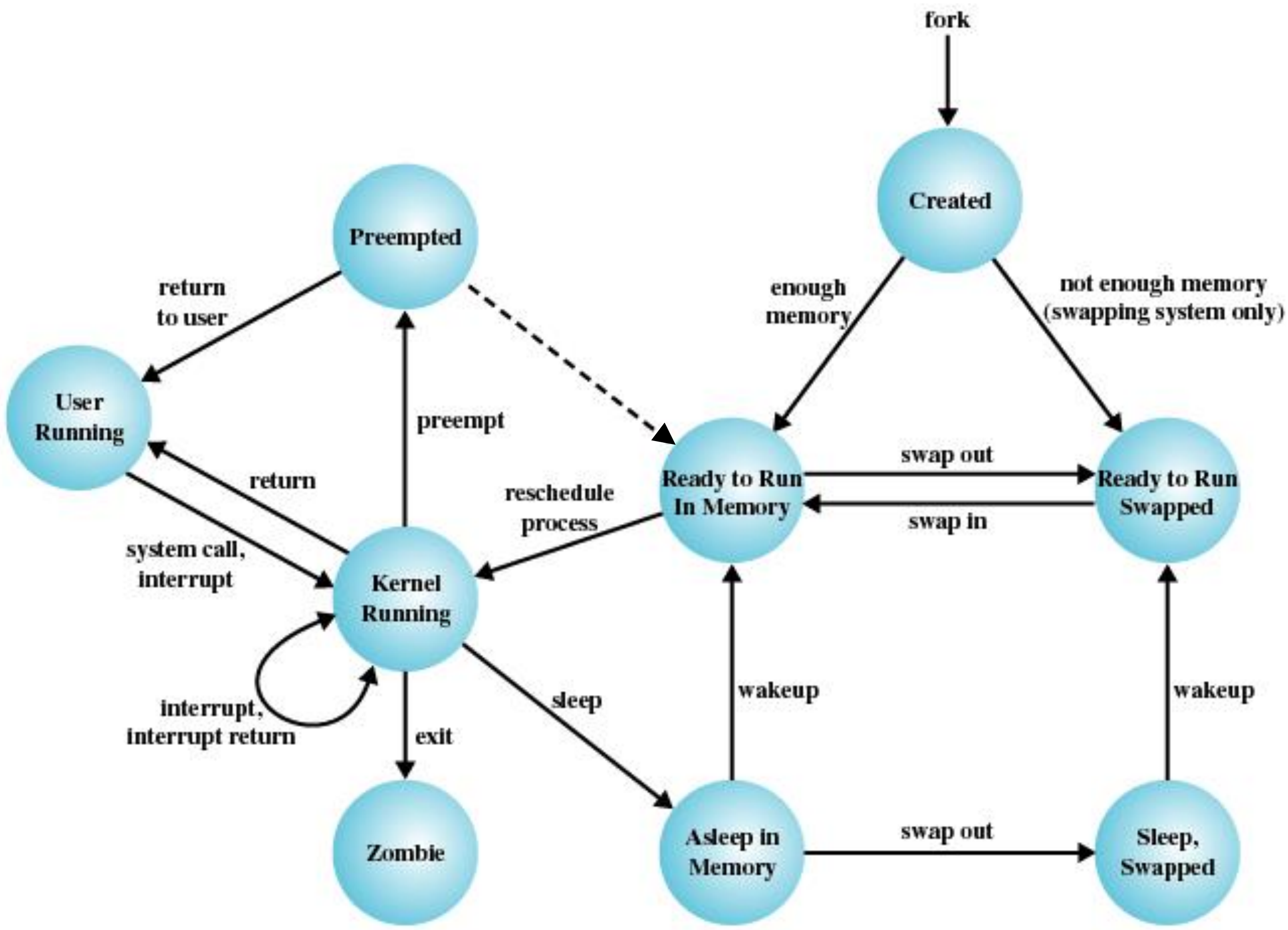
Example: UNIX – Architecture

- Process architecture that executes kernel functions in the context of a user process



- Two modes are used: user / kernel mode (Ring 3/Ring 0)
- Two types of processes: system / user processes
- ➢ System processes are implemented as part of kernel to run background services, e.g. swapping

Example: UNIX – Process State Diagram



Example: UNIX – Process States

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

Related System Calls

- **`int execve(const char *filename, char *const argv[], char *const envp[])`**
  - Executes program pointed to by **`filename`** with arguments **`argv`** and environment **`envp`** (in the form of key=value)
  - Effectively replaces the current program with another one
  - ➢ **`exec()`** family of library function
- **`pid_t fork(void)`**
  - Creates child process that differs from parent only in its PID (process identifier) and PPID (parent process identifier)
  - Returns 0 for child process and child's PID for parent process
- **`void _exit(int status)`**
  - Terminates calling process; closes open file descriptors; children are adopted by process 1; signals termination to parent
  - ➢ **`exit()`** library function
- **`pid_t wait(int *status)`**
  - Wait for state change in child of calling process

Programming Example

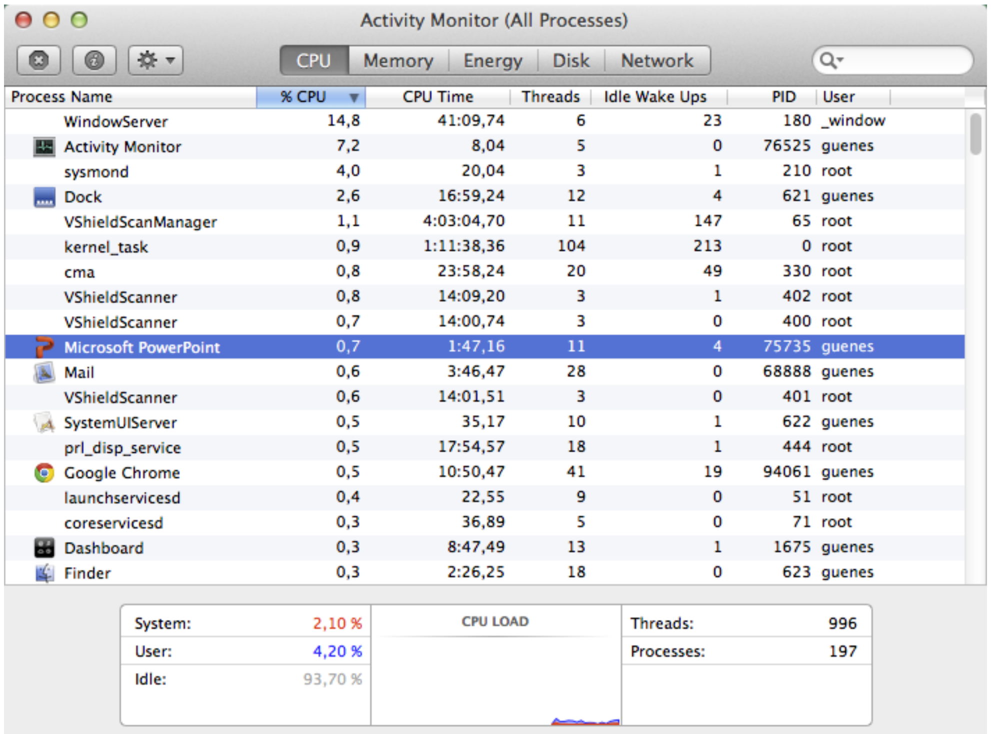

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>

main()
{
   int status;
   pid_t pid;

   pid = fork();
   if(pid == 0) {
     printf("Child process running...\n");
     // Do something...
     printf("Child process done.\n");
     exit(123);
   }
   else if(pid > 0) {
     printf("Parent process, waiting for child %d...\n", pid);
     pid = wait(&status);
     printf("Child process %d terminated, status %d.\n", pid, WEXITSTATUS(status));
     exit(EXIT_SUCCESS);
   }
   else {
     printf("fork() failed\n");
     exit(EXIT_FAILURE);
   }
}
```

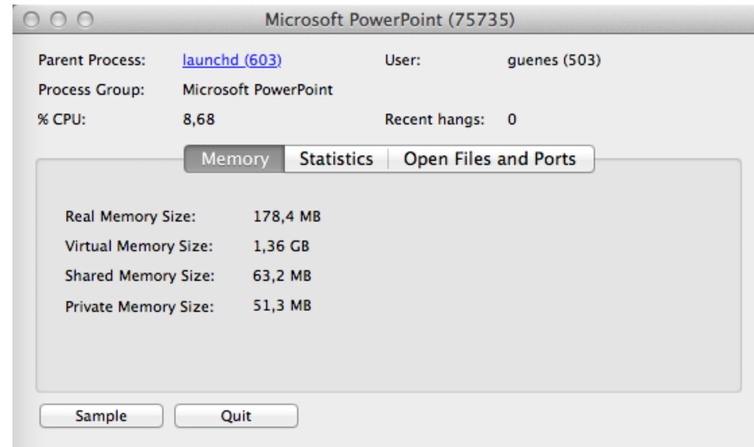
User-Level Process Control

wittenbu@vienna: /home/datsche/wittenbu - Shell - Konsole

```
wittenbu@vienna:~$ ps
  PID TTY          TIME CMD
19047 pts/1    00:00:00 csh
19050 pts/1    00:00:00 bash
19243 pts/1    00:00:00 ps
wittenbu@vienna:~$ kword &
[1] 19244
wittenbu@vienna:~$ ps
  PID TTY          TIME CMD
19047 pts/1    00:00:00 csh
19050 pts/1    00:00:00 bash
19244 pts/1    00:00:01 kword
19245 pts/1    00:00:00 ps
wittenbu@vienna:~$ kill -9 19244
wittenbu@vienna:~$ ps
  PID TTY          TIME CMD
19047 pts/1    00:00:00 csh
19050 pts/1    00:00:00 bash
19246 pts/1    00:00:00 ps
[1]+  Killed                  kword
wittenbu@vienna:~$
```

User-Level Process Control





Content (1)

1. Introduction and Motivation
2. Subsystems, Interrupts and System Calls
3. **Processes**
4. Memory
5. Scheduling
6. I/O and File System
7. Booting, Services, and Security