Introduction to System Programming

Process and Threads, Concurrency, Deadlock and Scheduling

Minal Shah

Outline

- Process Concept
- Process States
- Process Control
- System Calls for Process Management (fork, waitpid, exec Family),
- Process Scheduling:
  - Types and Algorithms,
- Introduction to Threads
- Principles of Concurrency,
  - Semaphores,
  - Monitors,
  - Reader/Writer Problem,

Outline

- System Calls for Semaphore Management (semget, semctl, semop, fsync)
- Deadlock:
  - Introduction,
  - Principles of Deadlock,
  - Deadlock Prevention,
  - Deadlock Avoidance,
  - Deadlock Detection

Large Building Project

Builder (General Contractor)

Operating System

Materials

Resources

Building

Process

Workers

Threads

Tools

Libraries

Identify

Computer
Data
Context
Programs
Interrupts

Scheduling

2 State
5 state

States

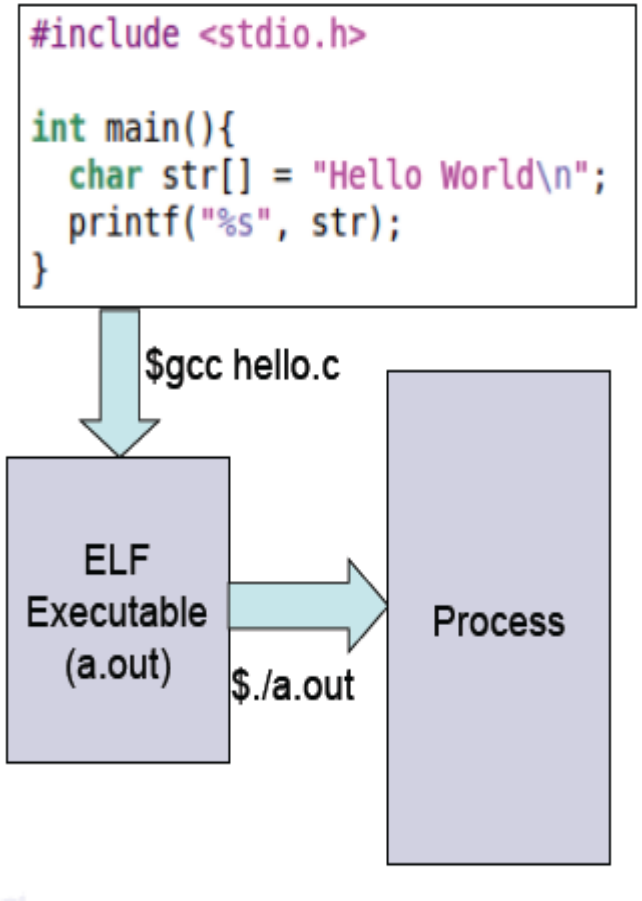
New
Working
Waiting
Blocked
Suspended
Completed

Questions…

- What is a process?
- When is a process created?
- When is a process terminated?
- What is a 2-state scheduling model?
- What additional states are added with a 5-state model?
- How does a suspended process differ from a blocked process?
- What task variables are found in a Task Control Table?

What is a “process”?

- *A program in execution*
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system instructions.
- Example :
- Single process **$ cat file1 file2 &**
- Two processes: $ **ls | wc - l**



OS Process Support?

- *Assist the execution of a process*
  - *interleave the execution of several processes*
  - *maximize processor utilization*
  - *provide reasonable response time*
- *Allocate resources to processes*
  - *fairness*
  - *avoid starvation / deadlock*
- *Support interprocess activities*
  - *communication*
  - *user creation of processes*

Process Elements

- A process is comprised of:
  - Program code (possibly shared)
  - A set of data
  - A number of attributes describing the state of the process
  - While the process is running it has a number of elements including : Identifier, State, Priority, Program counter, Memory pointers, Context data, I/O status information, Accounting information

Process Elements

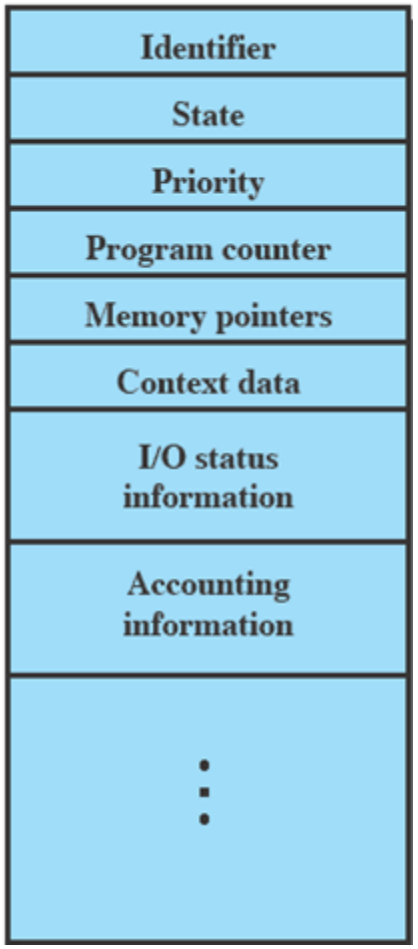
- **Identifier** : A unique identifier associated with this process , to distinguish it from all other processes.
- **State** : If the process is currently executing, it is in the running state. Other states may be ready, waiting.
- **Priority** : Priority level relative to other processes.
- **Program counter** : The address of the next instruction in the program to be executed.
- **Memory pointers** : Includes pointers to the program code and data associated with the process, plus and memory blocks shared with other process.

Process Elements

- **Context data** : These are data that are present in registers in the processor while the process is executing.
- **I/O status information** : Includes outstanding I/O requests, I/O devices (e.g., tape drives) assigned to this process, a list of files in use by the process, and so on.
- **Accounting information** : May include the amount of processor time and clock time used, time limits, account numbers, and so on.
- This information is stored in a data structure, typically called **a process control block**

Process Control Block

- Contains the process elements
- Created and manage by the operating system
- Allows support for multiple processes
- Process Control Block contains sufficient information so that it is possible to interrupt a running process and later resume execution as if the interruption had not occurred.

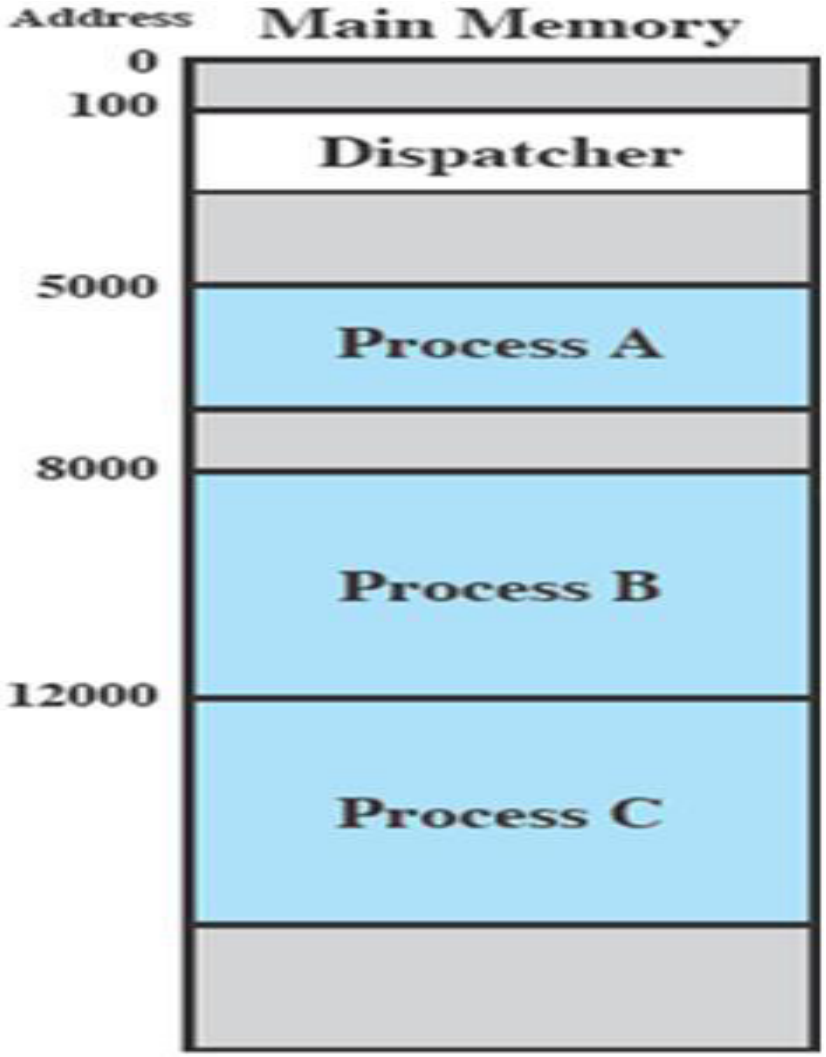


Simplified Process Control Block

Trace of the Process

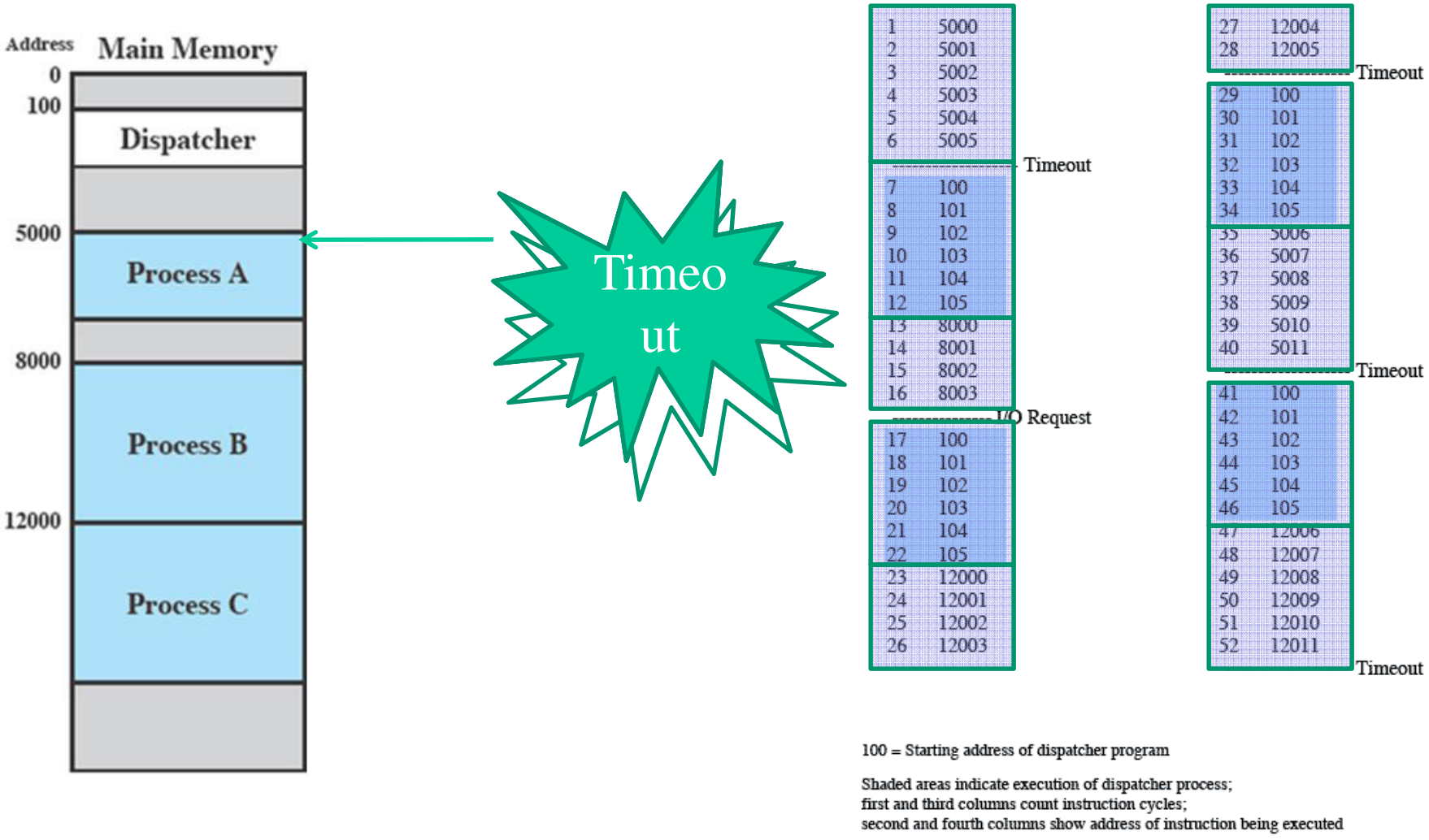
- The behavior of an individual process is shown by listing the sequence of instructions that are executed
- This list is called a Trace
- Dispatcher is a small program which switches the processor from one process to another

Process Execution



- Consider three processes being executed
- All are in memory (plus the dispatcher)
- Lets ignore virtual memory for this.

Trace from Processors point of view



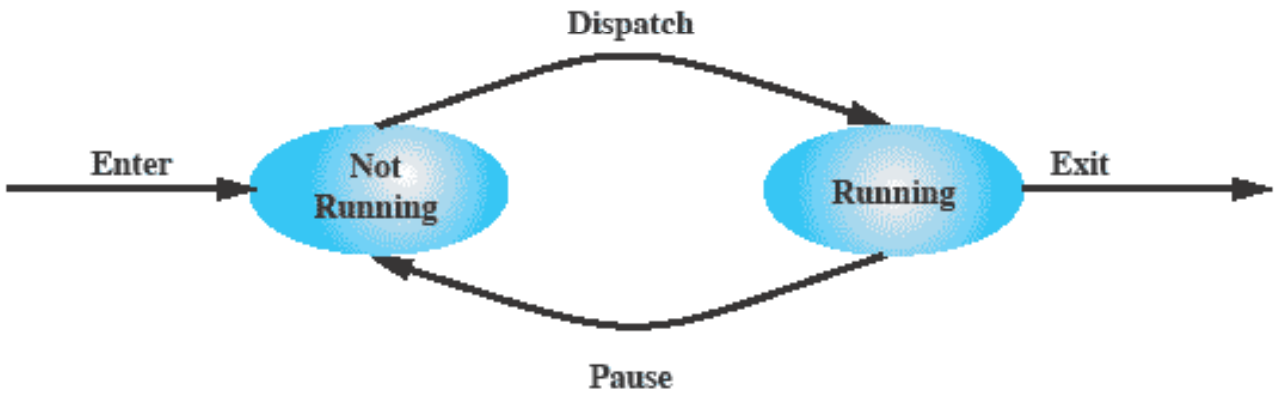
100 = Starting address of dispatcher program

Shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Figure 3.4 Combined Trace of Processes of Figure 3.2

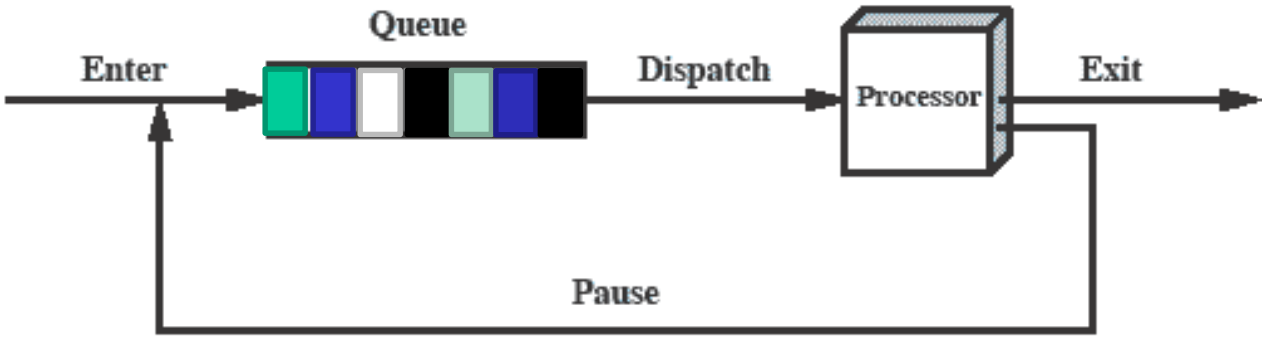
Two-State Process Model

- Process may be in one of two states
  - Running
  - Not-running



(a) State transition diagram

Queuing Diagram



(b) Queuing diagram

Etc … processes moved by the dispatcher of the OS to the CPU then back to the queue until the task is competed

Two-state Model Problems

- What does not-running mean?
  - Ready to execute?
  - Blocked?
  - Suspended?
  - Terminated?
  - Priority?
- Scheduler/dispatcher cannot just select the process that has been in the queue the longest.
- Challenge to make overall system as “efficient” and “fair” as possible.

When is a Process Created?

- Submission of a batch job
- User logs on
- Created to provide a service such as printing
- Process creates another process
  - Modularity
  - Parallelism
- Parent – child relationship
  - When the OS creates a process at the explicit request of another process, the action is referred to as **process spawning**. [**Parent Process** is the original, creating, process, **Child Process** is the new process]
- Deciding how to allocate the resources is a policy that is determined by the OS

Process Creation Decisions

- Resource Allocation
  - Treat as a new process
  - Divide parent's resources among children
- Execution
  - child runs concurrently with parent
  - parent waits until some or all children terminate
- Address Space
  - copy of parent
  - new program loaded into address space

Example: Unix Process Creation

- A new process is created by the fork call
- Child and parent are identical
  - child returns a 0
  - parent returns nonzero
- Both parent and child execute next line
- Often the child executes an exec
  - creates a brand new process in its space
- Parent can execute a wait

Unix Example

fork() returns:
-1 = error
0 = child
>0 = parent

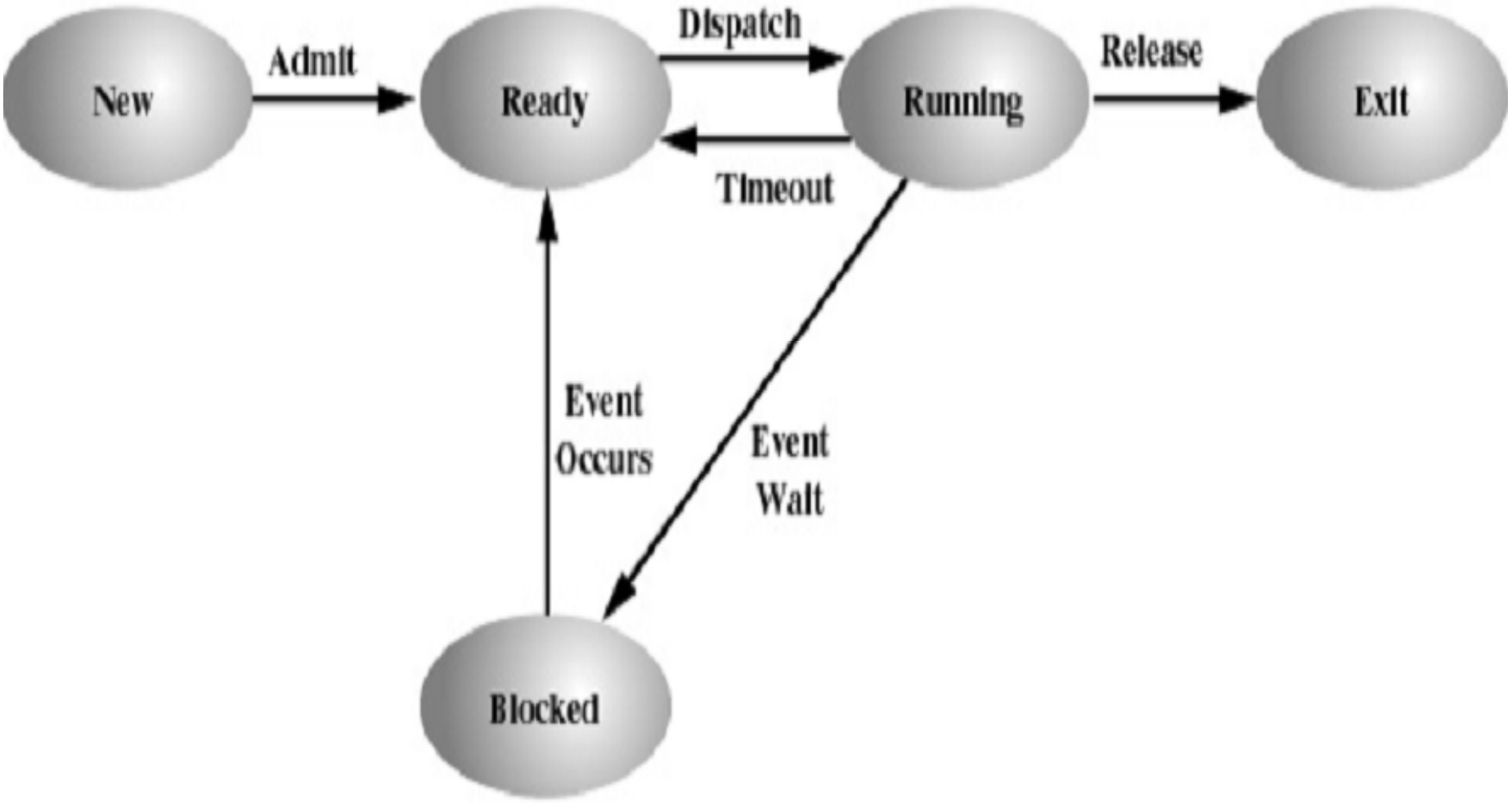
```
switch (child1 = fork())
{     case –1:printf("Can't fork");
                  exit(-1);
      case 0:  child1P(one2two, two2one);        // child 1
                  exit(-2);
      default: switch (child2 = fork())          // parent
                  {     case –1:printf("Can't fork");
                              exit(-1);
                        case 0:  child2P(one2two, two2one);
                              exit(-3);
                        default: // Wait for child two
                              waitpid(child2, status2, options);
                  }
                  /* Wait for child one */
                  waitpid(child1, status1, options);
                  fflush(stdout);
}
```

waitpid() joins parent & child

When is a Process Terminated?

- User logs off
- Normal completion
- Batch issues *Halt*
- Time limit exceeded
- Memory unavailable
- Bounds violation
- Protection error
  - example write to read-only file
- Arithmetic error
- Time overrun
  - event timeout
- I/O failure
- Invalid instruction
  - tried to execute data
- Privileged instruction
- Data misuse
- Operating system intervention
  - such as when deadlock occurs
- Parent terminates so child processes terminate
- Parent request

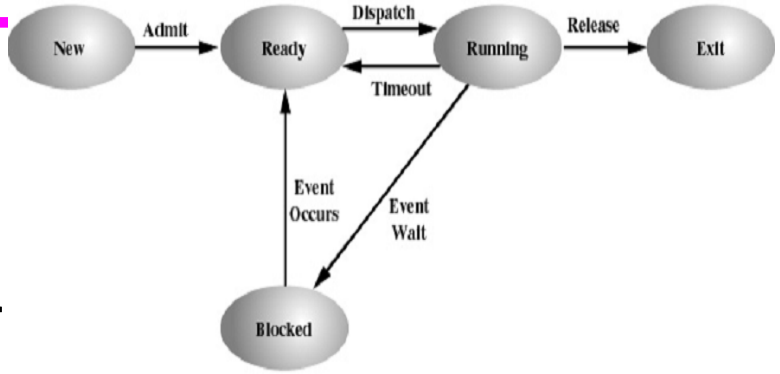
What is a 5-State Scheduling Model?



5-State Scheduling Model

- **Running**: The process that is currently being executed. with a single processor, so at most one process at a time can be in this state.
- **Ready**: A process that is prepared to execute when given the opportunity.
- **Blocked/Waiting**: A process that cannot execute until some event occurs, such as the completion of an I/O operation.
- **New**: A process that has just been created but has not yet been admitted to the pool of executable processes by the OS. Typically, a new process has not yet been loaded into main memory, although its process control block has been created.
- **Exit:** A process that has been released from the pool of executable processes by the OS, either because it halted or because it aborted for some reason.

Five-state Model Transitions

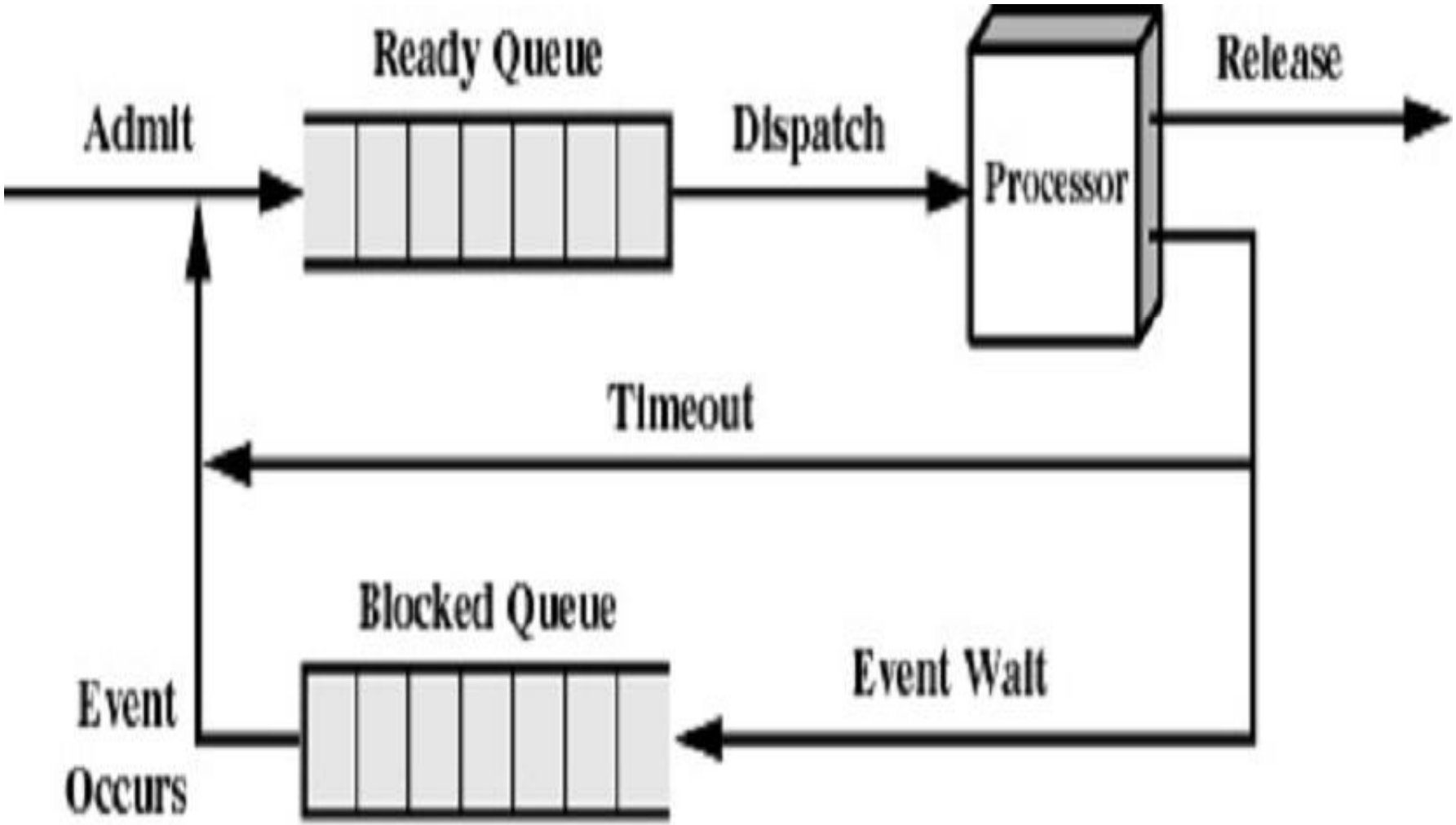


- **Null → New:** Process is created
- **New → Ready:** O.S. is ready to handle another process **(Admit)**
- **Ready → Running:** Select another process to run **(Dispatch)**
- **Running → Exit:** Process has terminated **(Release)**
- **Running → Ready:** End of time slice or higher-priority process is ready **(Timeout)**
- **Running → Blocked:** Process waiting for event **(Event Wait)**
- **Blocked → Ready:** The event a process is waiting for has occurred, can continue **(Event Occurs)**
- **Ready → Exit:** Process terminated by O.S. or parent
- **Blocked → Exit:** Same reasons

Five-state Model Transitions

- If all processes were always ready to execute, then the simple FIFO queuing model would suffice, but processor operates in round- robin fashion on available processes.

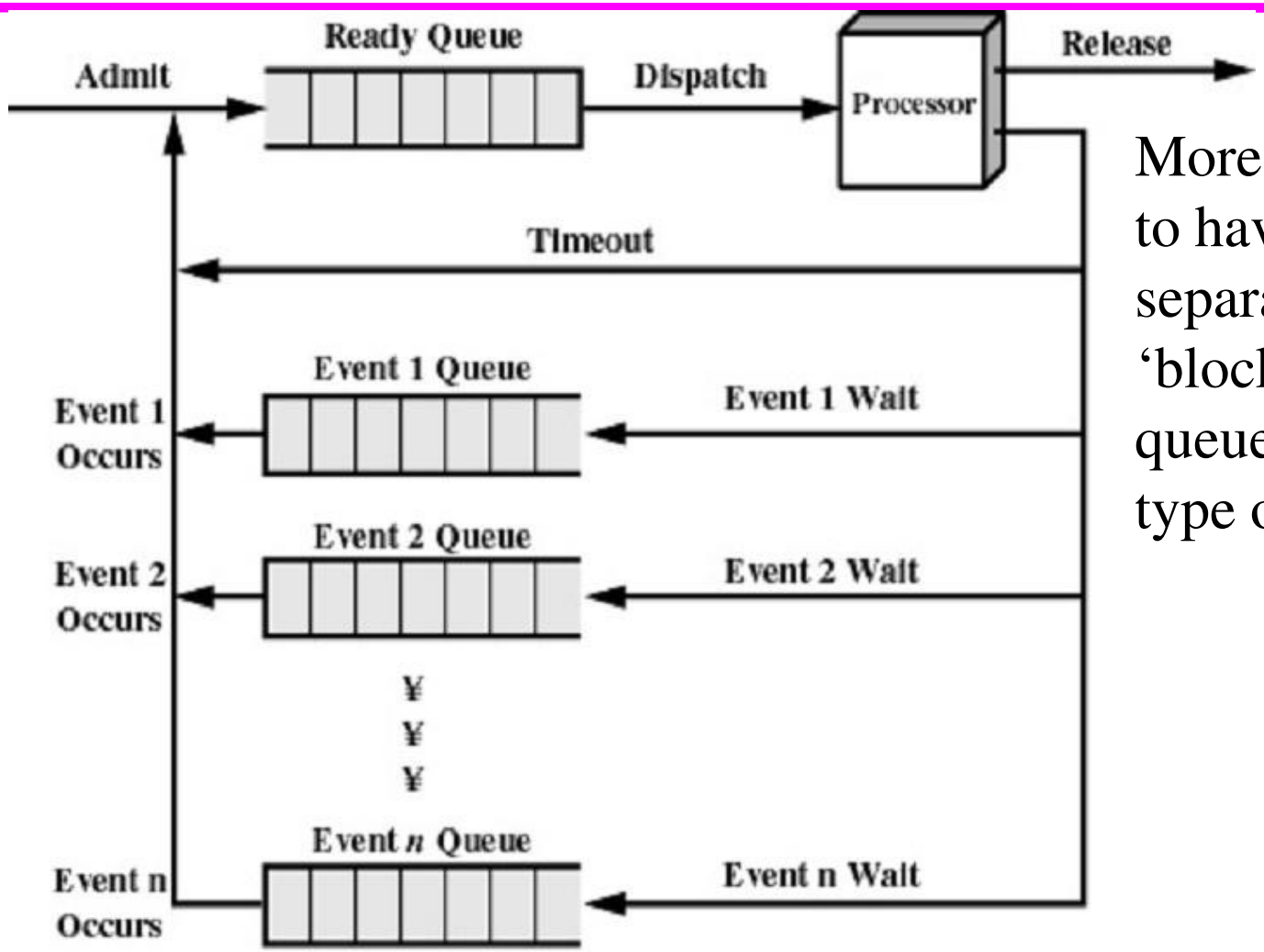
What is a 5-State Scheduling Model? (single blocked queue)



Using Two Queues

- In the simplest solution, this model would require an additional queue for the blocked processes.
- But when an event occurs the dispatcher would have to cycle through the entire queue to see which process is waiting for the event.
- This can cause huge overhead when their may be 100’s or 1000’s of processes

Multiple Blocked Queues

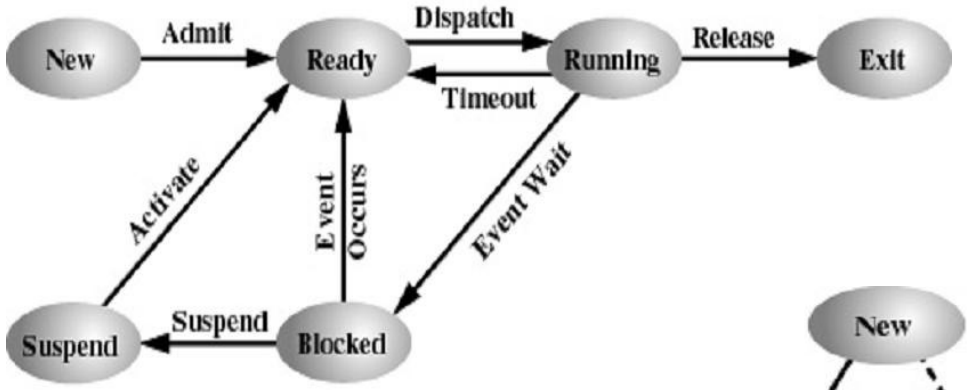


More efficient to have a separate 'blocked' queue for each type of event.

What is a Suspended Process?

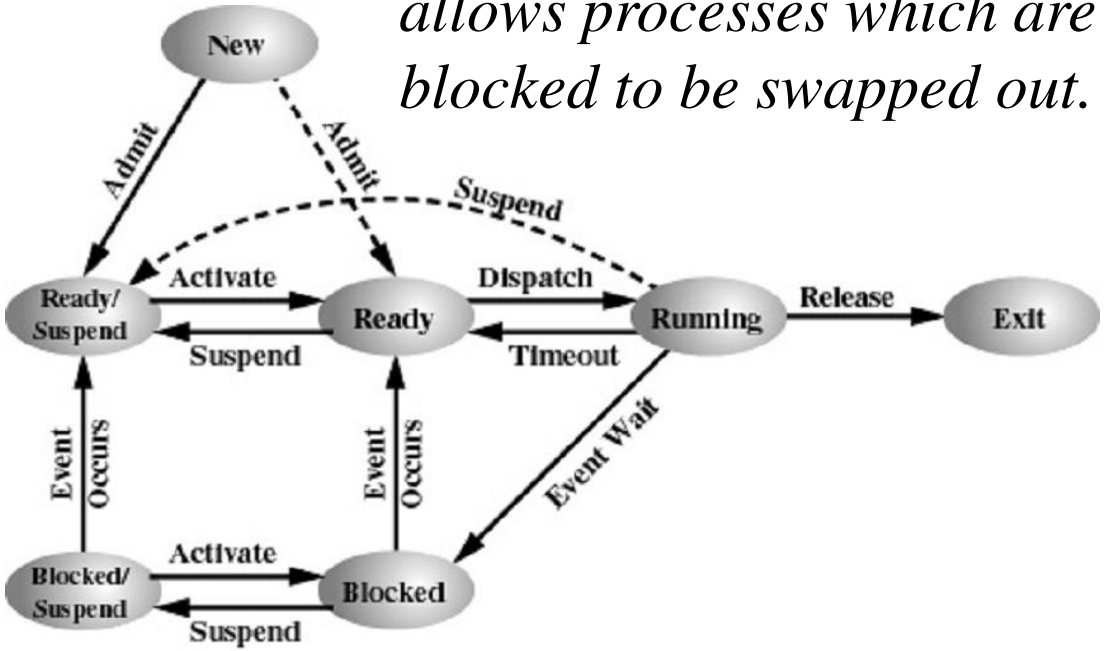
- Processor is faster than I/O so all processes could be waiting for I/O
- Swap these processes to disk to free up more memory
- Blocked state becomes **suspende**d state when swapped to disk
- Two new states
  - Blocked, suspend
  - Ready, suspend

Suspended State Scheduling



One Suspend State

the simple solution is to add a single state – but this only allows processes which are blocked to be swapped out.



Two Suspend State

Two suspend states allow all processes which are not actually running to be swapped.

Reasons for States of Process

- There are two independent concepts here: whether a process is waiting on an event (blocked or not)
- And whether a process has been swapped out of main memory (suspended or not).
- To accommodate this 2 x 2 combination, we need four states:
  - Ready: The process is in main memory and available for execution.
  - Blocked: The process is in main memory and awaiting an event.
  - Blocked/Suspend: The process is in secondary memory and awaiting an event.
  - Ready/Suspend: The process is in secondary memory but is available for execution as soon as it is loaded into main memory.

Transitions States

- **Blocked →Blocked/Suspend:** If there are no ready processes, then at least one blocked process is swapped out to make room for another process that is not blocked.
  - This transition can be made even if there are ready processes available, if the OS determines that the currently running process or a ready process that it would like to dispatch requires more main memory to maintain adequate performance.
- **Blocked/Suspend → Ready/Suspend:** A process in the Blocked/Suspend state is moved to the Ready/Suspend state when the event for which it has been waiting occurs.

Transitions States

- **Ready/Suspend → Ready:** When there are no ready processes in main memory**,** the OS will need to bring one in to continue execution. In addition, it might be the case that a process in the Ready/Suspend state has higher priority than any of the processes in the Ready state. In that case, the OS designer may dictate that it is more important to get at the higher-priority process than to minimize swapping.

Transitions States

- **Ready → Ready/Suspend:** OS would prefer to suspend a blocked process rather than a ready one, because the ready process can now be executed, whereas the blocked process is taking up main memory space and cannot be executed.
  - However, it may be necessary to suspend a ready process if that is the only way to free up a sufficiently large block of main memory.
  - OS may choose to suspend a lower-priority ready process rather than a higher priority blocked process if it believes that the blocked process will be ready soon.

Transitions States

- **New → Ready/Suspend and New → Ready:** When a new process is created, it can either be added to the Ready queue or the Ready/Suspend queue.
  - OS must create a process control block and allocate an address space to the process. It might be preferable for the OS to perform these housekeeping duties at an early time, so that it can maintain a large pool of processes that are not blocked.
  - With this strategy, there would often be insufficient room in main memory for a new process; hence the use of the (New → Ready/Suspend) transition.

Transitions States

- **Blocked/Suspend → Blocked:** Inclusion of this transition may seem to be poor design. After all, if a process is not ready to execute and is not already in main memory, what is the point of bringing it in?
  - But consider the following scenario: A process terminates, freeing up some main memory. There is a process in the (Blocked/Suspend) queue with a higher priority than any of the processes in the (Ready/Suspend) queue and the OS has reason to believe that the blocking event for that process will occur soon.
  - Under these circumstances, it would seem reasonable to bring a blocked process into main memory in preference to a ready process.

Transitions States

- **Running → Ready/Suspend:** Normally, a running process is moved to the Ready state when its time allocation expires. If, however, the OS is preempting the process because a higher-priority process on the Blocked/Suspend queue has just become unblocked, the OS could move the running process directly to the (Ready/Suspend) queue and free some main memory.
- **Any State → Exit:** Typically, a process terminates while it is running, either because it has completed or because of some fatal fault condition. OS, a process may be terminated by the process that created it or when the parent process is itself terminated. If this is allowed, then a process in any state can be moved to the Exit state.

Reasons for Process Suspension

- **Swapping :** The OS needs to release sufficient main memory to bring in a process that is ready to execute
- **Other OS reason :** The OS may suspend a background or utility process or a process that is suspected of causing a problem
- **Interactive user request :** A user may wish to suspend execution of a program for purposes of debugging or in connection with the use of a resource
- **Timing :** A process may be executed periodically and may be suspended while waiting for the next time interval
- **Parent process request :** A parent process may wish to suspend execution of a descendent to examine or modify the suspended process

Suspended Process Characteristics

- The process is not immediately available for execution.
- The process may or may not be waiting on an event. If it is, this blocked condition is independent of the suspend condition, and occurrence of the blocking event does not enable the process to be executed immediately.
- The process was placed in a suspended state by an agent: either itself, a parent process, or the OS, for the purpose of preventing its execution.
- The process may not be removed from this state until the agent explicitly orders the removal.

Process Control
[Modes of Execution]

- Most processors support at least two modes of execution
- User mode
  - Less-privileged mode
  - User programs typically execute in this mode
- System mode
  - Kernel of the operating system
  - More-privileged mode
    - Including reading or altering a control register, such as the program status word;
    - primitive I/O instructions;
    - Instructions that relate to memory management.

Process Control
[Modes of Execution]

- Questions: How does the processor know in which mode it is to be executing? And how does it change
- Answer: Typically a flag (single bit) in the program status word (PSW). This bit is changed in response to certain events. Typically, when a user makes a call to an operating system service or when an interrupt triggers execution of an operating system routine, the mode is set to the kernel mode and, upon return from the service to the user process, the mode is set to user mode.

Process Control
[Typical Functions of an OS Kernel]

- Process Management
  - Process creation and termination
  - Process scheduling and dispatching
  - Process switching
  - Process synchronization and support for interprocess communication
  - Management of process control blocks
- Memory Management
  - Allocation of address space to processes
  - Swapping
  - Page and segment management

Process Control
[Typical Functions of an OS Kernel]

- I/O Management
  - Buffer management
  - Allocation of I/O channels and devices to processes
- Support Functions
  - Interrupt handling
  - Accounting
  - Monitoring

Process Control
[Process Creation]

- Once the OS decides to create a new process it:
  - Assigns a unique process identifier
  - Allocates space for the process
  - Initializes process control block
  - Sets up appropriate linkages
  - Creates or expand other data structures

Process Control
[Switching Process]

- Several design issues are raised regarding process switching
  - What events trigger a process switch?
  - We must distinguish between mode switching and process switching.
  - What must the OS do to the various data structures under its control to achieve a process switch?

Process Control
[When to switch processes]

- A process switch may occur any time that the OS has gained control from the currently running process.

Mechanism	Cause	Use
Interrupt	External to the execution of the current instruction	Reaction to an asynchronous external event
Trap	Associated with the execution of the current instruction	Handling of an error or an exception condition
Supervisor call	Explicit request	Call to an operating system function

Process Control
[When to switch processes]

- Two kinds of system interrupts,
  - one is simply called an interrupt,
  - and the other called a trap.
- “interrupts” are due to some sort of event that is external to and independent of the currently running process, such as the completion of an I/O operation.
- With an ordinary interrupt, control is first transferred to an interrupt handler, which does some basic housekeeping and then branches to an OS routine that is concerned with the particular type of interrupt that has occurred.

Process Control
[When to switch processes]

- "Traps" relate to an error or exception condition generated within the currently running process, such as an illegal file access attempt.
  - With traps, the OS determines if the error or exception condition is fatal.
  - If so, then the currently running process is moved to the Exit state and a process switch occurs.
  - If not, then the action of the OS will depend on the nature of the error and the design of the OS.
    - It may attempt some recovery procedure or simply notify the user.
    - It may do a process switch or resume the currently running process.

Process Control
[When to switch processes]

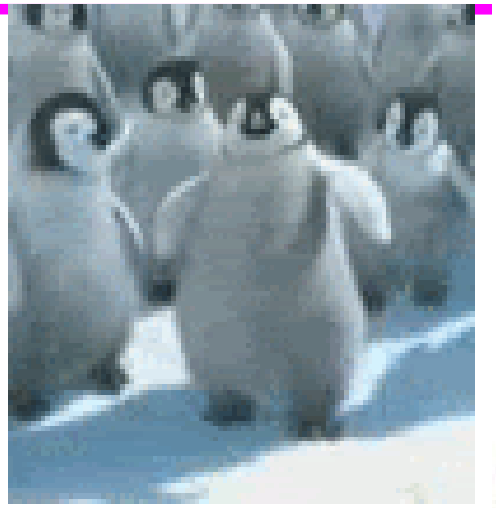
- Finally, the OS may be activated by a supervisor call from the program being executed.
  - For example, a user process is running and an instruction is executed that requests an I/O operation, such as a file open.
- This call results in a transfer to a routine that is part of the operating system code. The use of a system call may place the user process in the Blocked state.

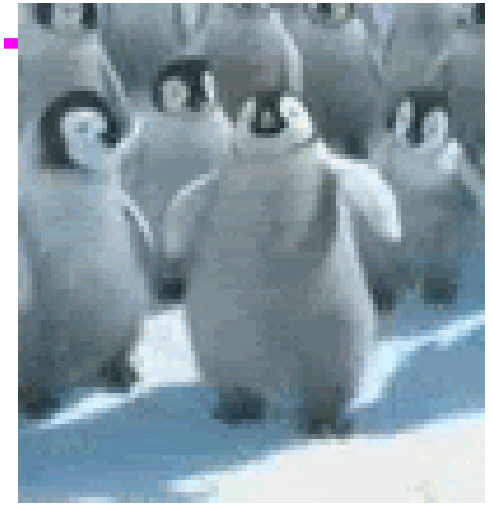
Process Control
[Change of Process State]

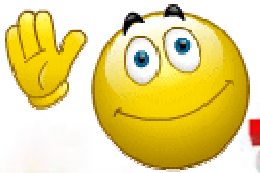
- The steps involved in a full process switch are as follows :
  1. Save context of processor including program counter and other registers
  2. Update the process control block of the process that is currently in the Running state
     - This includes changing the state of the process to one of the other states (Ready; Blocked; Ready/Suspend; or Exit). Other relevant fields must also be updated, including the reason for leaving the Running state and accounting information.

Process Control
[Change of Process State]

3. Move process control block to appropriate queue – ready; blocked; ready/suspend
4. Select another process for execution
5. Update the process control block of the process selected
   - This includes changing the state of this process to Running
6. Update memory-management data structures
   - This may be required, depending on how address translation is managed;
7. Restore context of the selected process







Minal Shah