Ollscoil na Gaillimhe
University of Galway

Dr Takfarinas Saber
takfarinas.saber@universityofgalway.ie

CT213
Computing Systems & Organisation

Process Management

Content

1. Process Manager Process – User perspective
2. Process – Operating System perspective
3. Threads
4. Operating System services for process management

Program and Process

- **Program**: static entity made up of source program language statements that define process behavior when executed on a set of data
- **Process**: dynamic entity that executes a program on a particular set of data using resources allocated by the system
  - two or more processes could execute the same program, each using its own data and resources

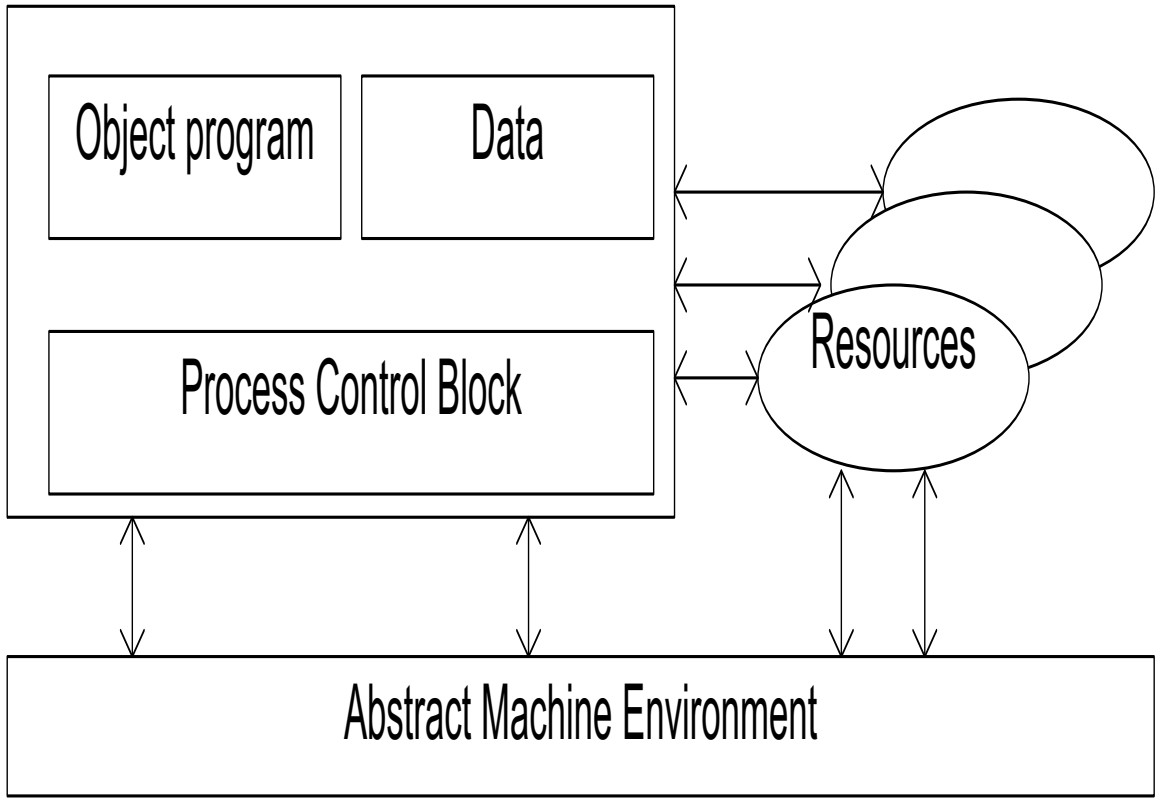
What is a Process?

- A process is a program in execution

- It is composed of:
  - Program
  - Data
  - Process Control Block (PCB): contains the state of the process in execution
    - What it is?
    - How much of its processing has been completed?
    - Etc.

Execution of a Process

- In order to execute, a process needs an **Abstract Machine Environment** to manage its use of resources
- Process Control Block (PCB) is required to map the environment state on the physical machine state
- The OS keeps a *process descriptor* for each process



Program Execution

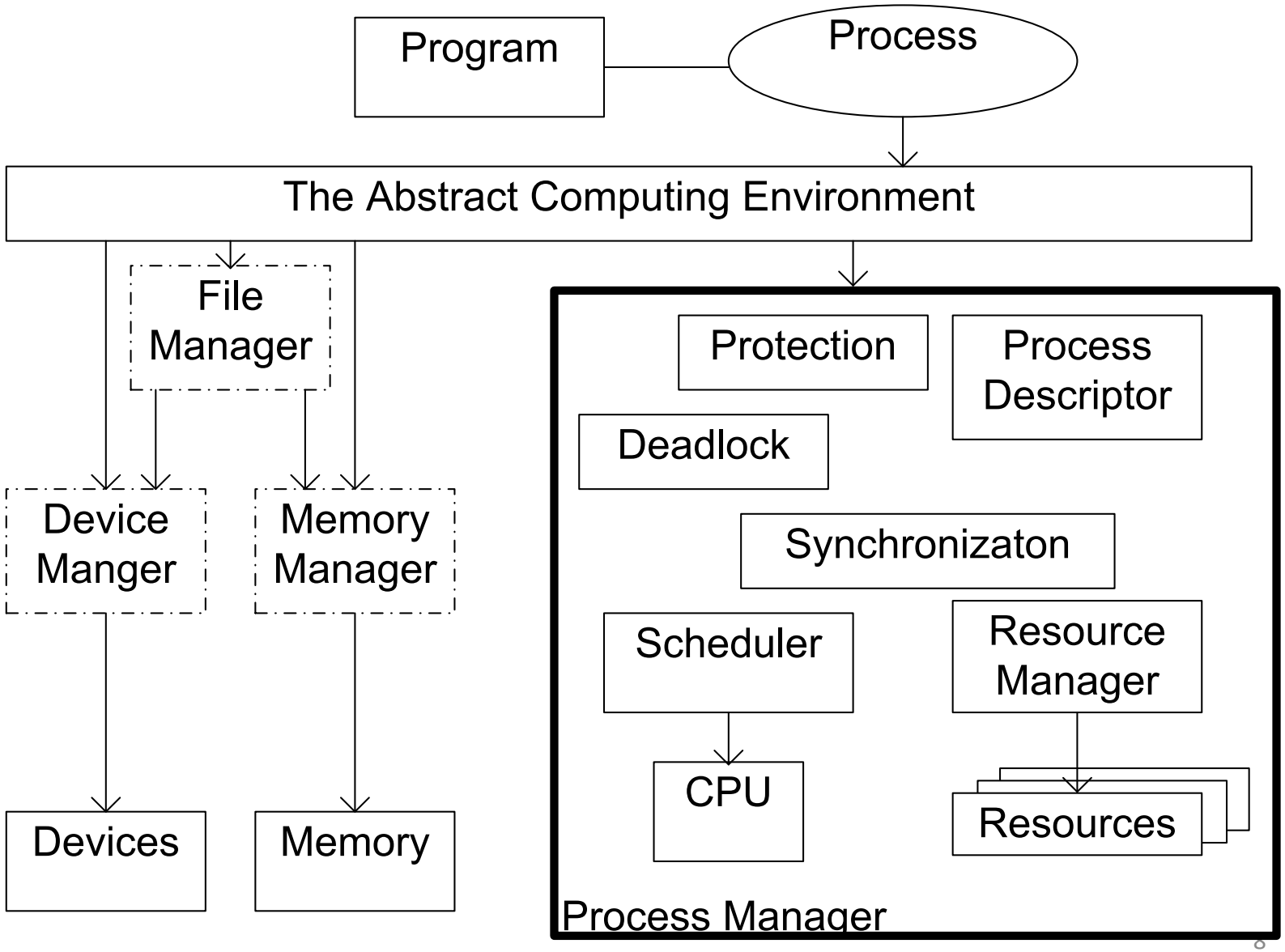
- Each execution of the program generates a process that is executed

- Inter-process relationships:
  - **Competition** – processes are trying to get access to different resources of the system, therefore a protection between processes is necessary
  - **Cooperation** – sometime the processes need to communicate between themselves and exchange information – synchronization is needed

Process Manager Functions

- Implements:
  1. CPU sharing (called *scheduling*)
     - allocate resources to processes in conformance with certain policies
  2. Process synchronization and inter-process communication
     - deadlock strategies and protection mechanisms

Process Manager



Process – User Perspective

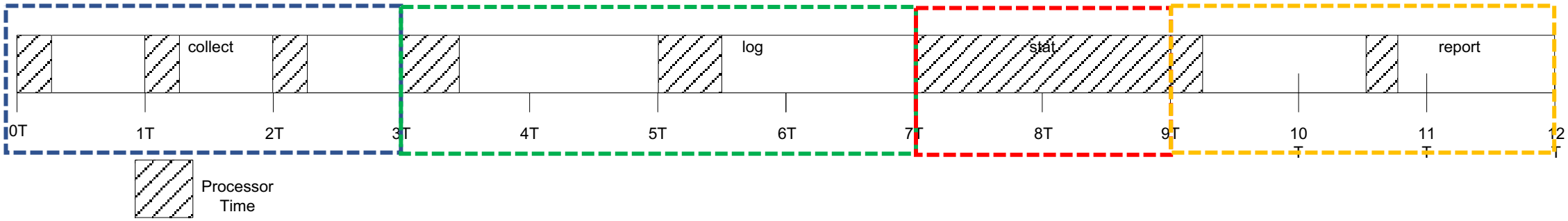
Example:

Consider an application that monitors an industrial process, to record its operation.

- The application contains 4 program modules:
  - **Data acquisition (*collect*):** Reads 3 values from a converter
    - Collecting each data is 1 T interval: ¼ T processor time, ¾ T is wait time to finish read op from converter
  - **Data storage (*log*):** Writes on the disk the 3 values read by *collect*:
    - It needs two operations (writing two values at a time, including “newline:) which take 2 T intervals each: 2/4 T processor time and 6/4 T wait time to finish the write operation
  - **Statistical processing (*stat*):** Statistical processing of the three values collected by *collect*, needs 2T
  - **Print results (*report*):** Prints two values resulted from statistical processing (*stat*)
    - Each print operation requires ¼ T processor time and 5/4 T wait time to finish print operation

Sequential Implementation

```
main(){
  while (TRUE){
      collect();
      log();
      stat();
      report();
       }
   }
```



- The time required for a cycle is 12T:
  - 4.25T is required for processing time (processor time)
  - 7.75T is wait time between various I/O operations

Multitasking Implementation

- The following processes will be executed in a quasi-parallel fashion, with the following priorities:
  - Log
  - Collect
  - Report
  - Stat

- For correct functionality, the processes need to synchronize to each other; this will be done with directives wait/signal:
  - **Wait** – wait for a signal from a specific process
  - **Signal** – send a signal to a specific process

```
void log(){
   while(TRUE){
       wait(collect);
       log_disk();
       signal (collect);
   }
}
```

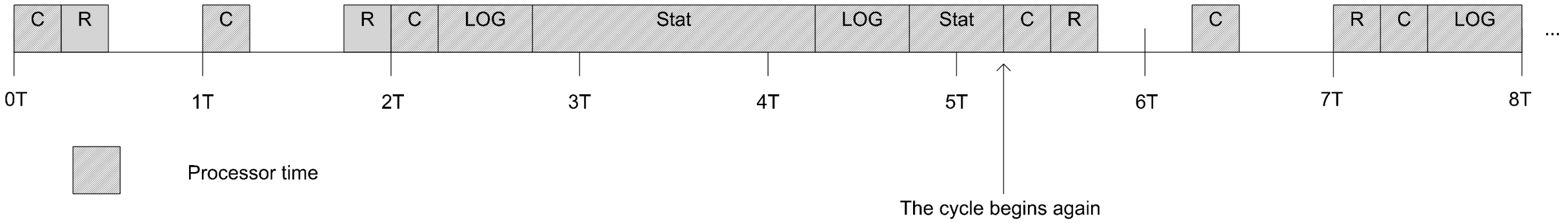
```
void report(){
   while (TRUE){
       wait (stat);
       report_pr ();
       signal (stat);
   }
}
```

```
void collect(){
   while(TRUE){
      wait (log);
      wait (stat);
      collect_ad ();
      signal (log);
      signal (stat);
   }
}
```

```
void stat(){
   while (TRUE){
       wait (collect);
       wait (report);
       stat_ad ();
       signal (collect);
       signal (report)
   }
}
```

```
main(){
  init_proc(&log(), …);
  init_proc(&collect(), …);
  init_proc(&report(), …);
  init_proc(&stat(),…);

  signal (collect); signal (collect);
  signal (stat);
  start_schedule();
}
```



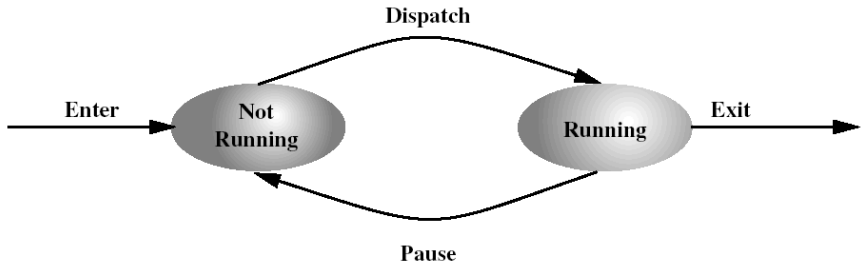
- 5.25 T for the execution of a complete cycle
- Only 1T lost in waiting time between I/O operations

Process – OS Perspective

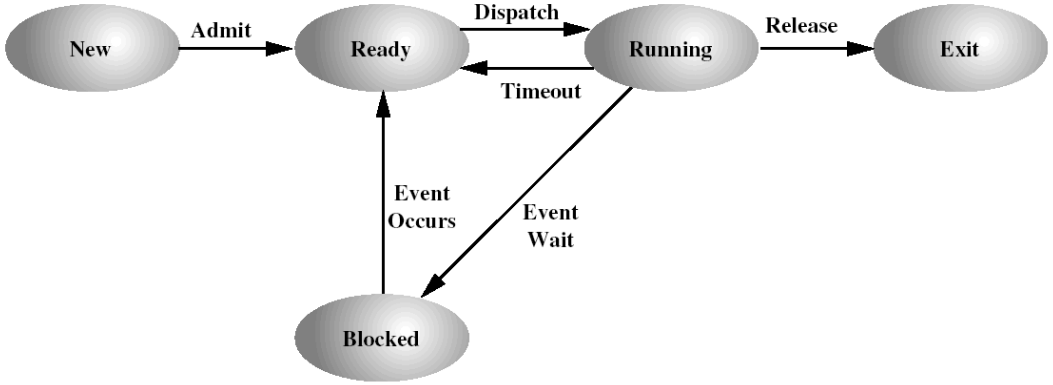
- The processor's principal function: execute machine instructions residing in main memory
  - Those instructions are provided in the form of programs
  - A processor may interleave the execution of a number of programs over time
- **Program View**
  - Its execution involves a sequence of instructions within that program
  - The behavior of individual process can be characterised by a sequence of instructions
    - ***trace*** of the process
- **Processor View**
  - Executes instructions from main memory, as dictated by changing values in the program counter (PC) register
  - The behaviour of the processor can be characterised by showing how the traces of various processes are **interleaved**

State Process Models

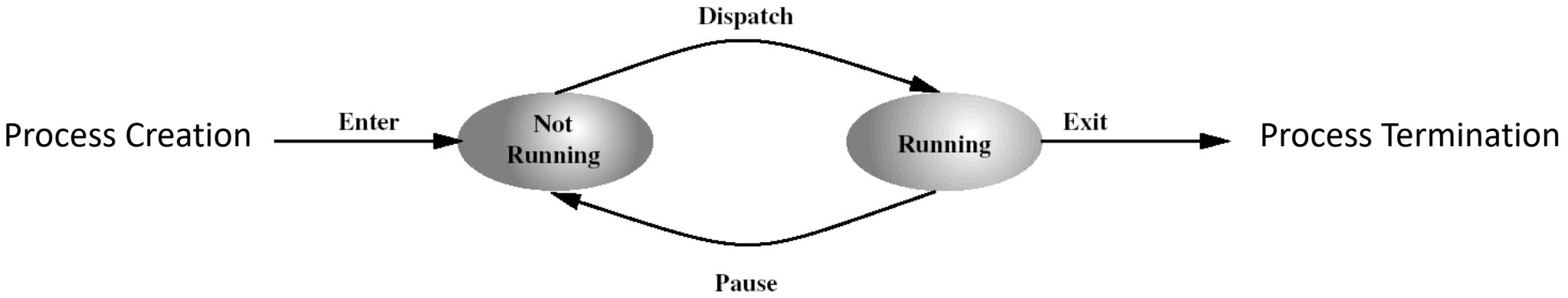
- Two State Process Model



- Five State Process Model



Two State Model



- The process can be in one of two states:
  - running or not running
- When the OS creates a new process, it enters it into the ***Not Running*** state; after that, the process exists, is known to the OS and waits for the opportunity to run
- From time to time, the currently running process will be interrupted and the dispatcher process will select a new process to run
  - The new process will be moved to **Running** state and the former one to ***Not Running*** state

Process Creation

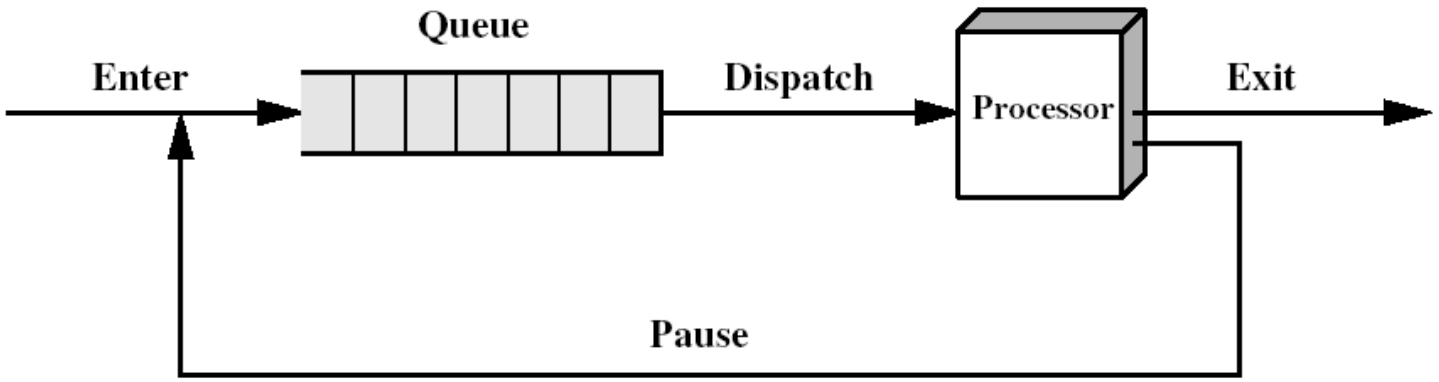
- Creation of new process:
  - The OS builds the data structures that are used to manage the process
  - The OS allocates space in main memory to the process
- Reasons for process creation:
  - New batch job
  - Interactive logon
  - Created by OS to provide a service
    - i.e., process to control printing
  - Spawned by existing process
    - i.e., to exploit parallelism

Process Termination

- Reasons for process termination
  - Process finished its execution (natural completion)
  - Total time limit exceeded
  - Errors (memory unavailable, arithmetic error, protection error, invalid instruction, privileged instruction, I/O failure, etc.)
  - Parent request
    - A parent has typically the right to request a child termination
  - Parent termination
    - When the parent terminates, the OS may automatically terminate all of its children

Queuing Discipline

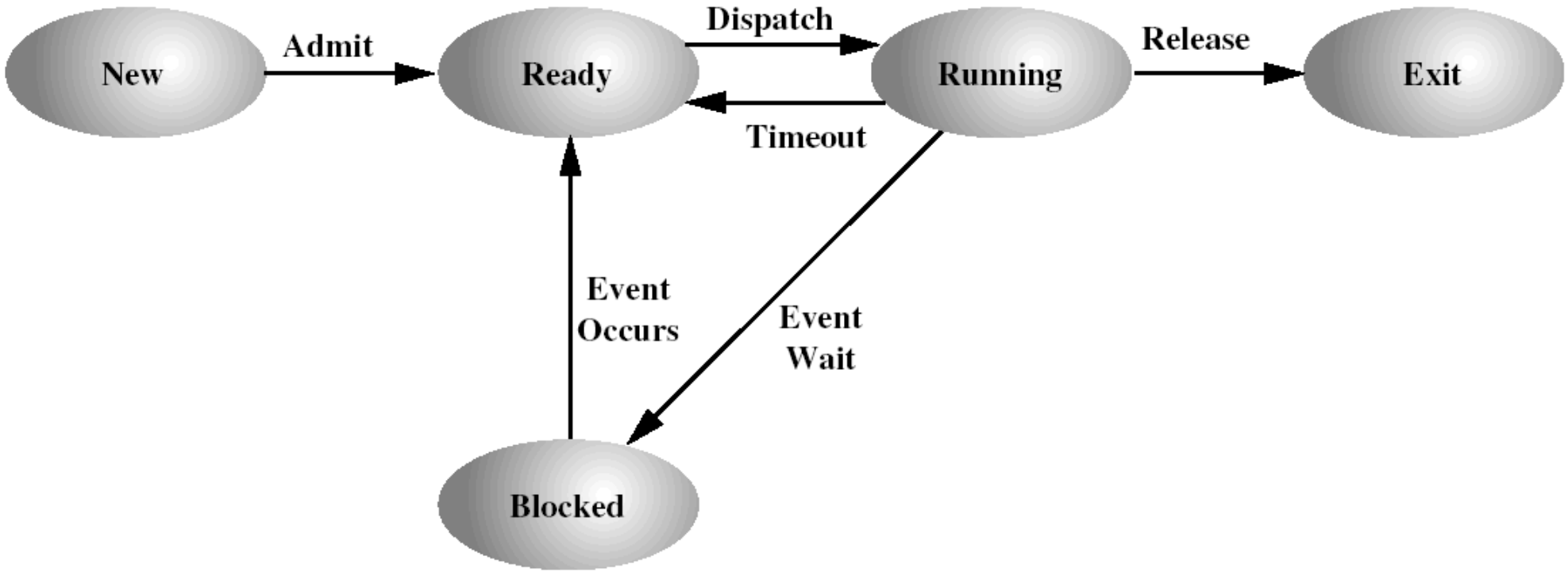
- Each process needs to be represented
  - Info relating to each process, including current state and location in memory
  - Waiting processes should be kept in some sort of queue
    - List of pointers to processes blocks
    - Linked list of data blocks; each block representing a process
- Dispatcher behavior:
  - An interrupted process is transferred in waiting queue
    - If process is completed or aborted, it is discarded
  - The dispatcher selects a process from the queue to execute



Five State Model

- ***Running*** – The process is currently being executed
  - For single processor systems, one single process can be in this state at a time
- ***Ready*** – a process that is prepared to execute when given the turn
- ***Blocked*** – a process that can't execute until some event occurs
  - Such as the completion of an I/O operation
- ***New*** – a process that has been created, but not yet accepted in the pool of executable processes by OS
  - Typically, a new process has not yet been loaded into main memory
- ***Exit*** – a process that has been released from the pool of executable processes by the OS
  - Completed or due to some errors

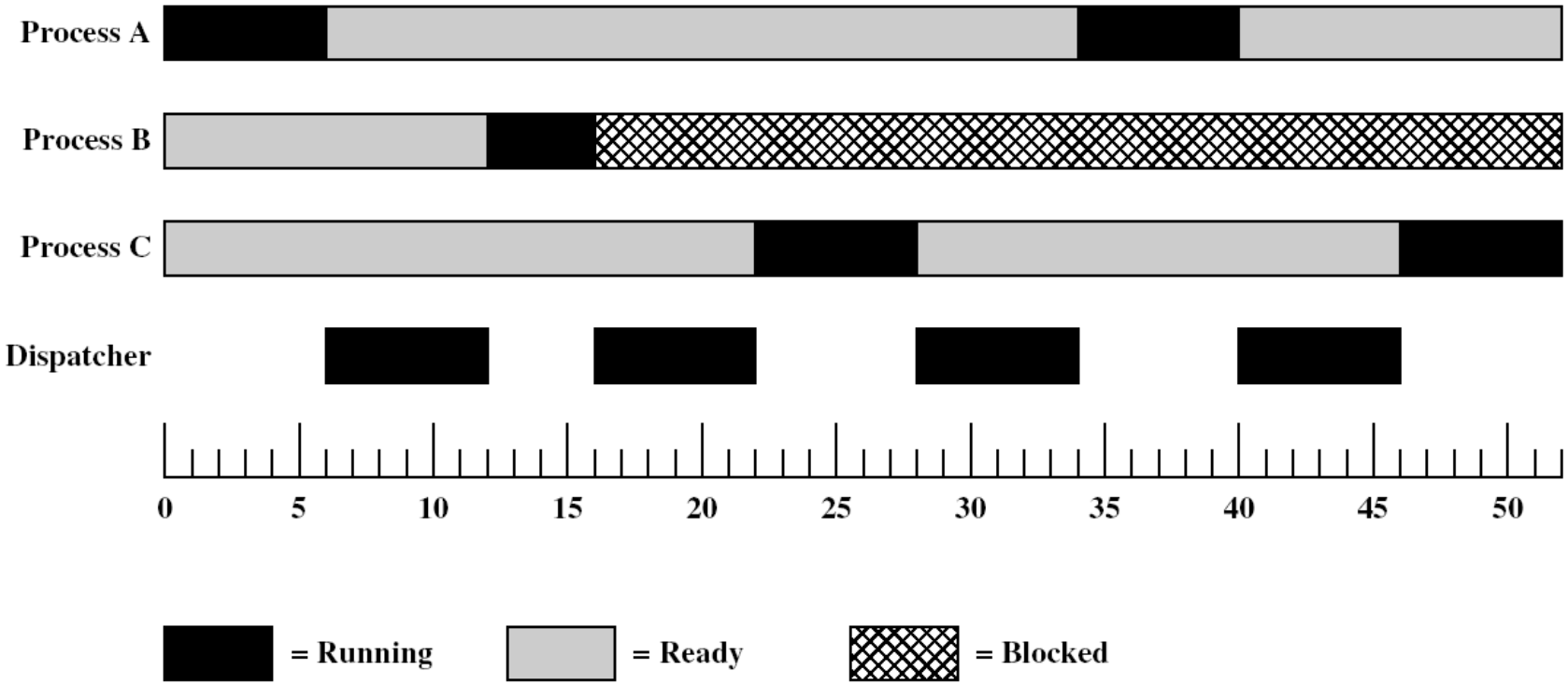
Five State Model Process Transition Diagram



Process – OS Perspective

- Consider three processes: A, B and C that are loaded in memory
- In addition, there is a small dispatcher program that switches the processor from one process to another (using Round Robin with 6 instructions)
- No use of virtual memory
- Process B invokes an I/O operation in its fourth instruction

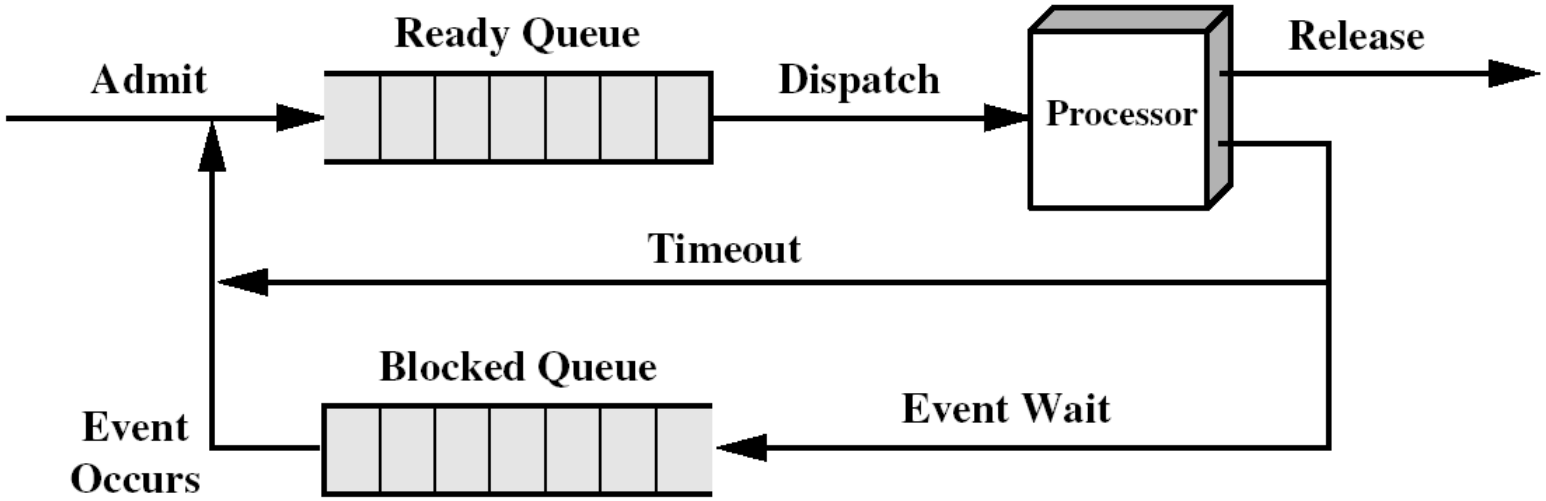
Example



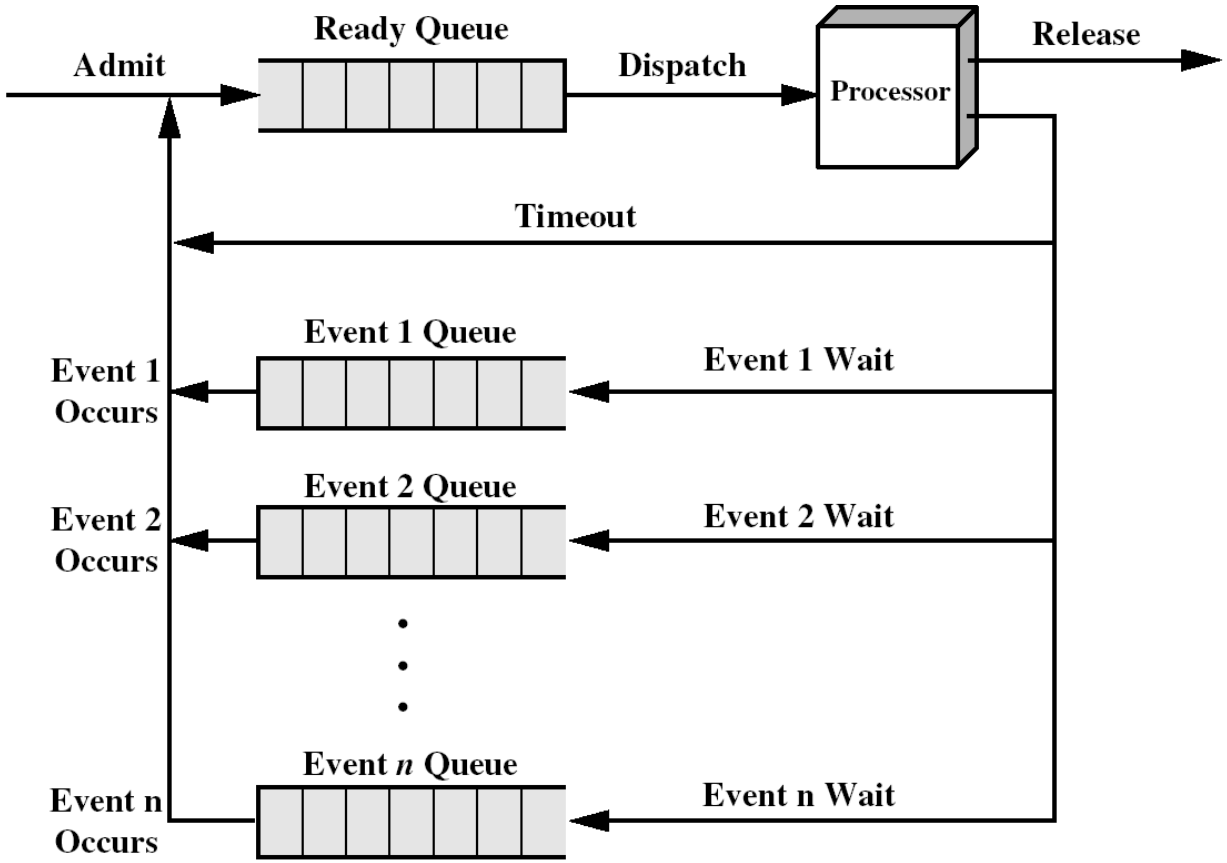
- Movement of each process described earlier (A, B and C) among the states

Queuing Discipline (1)

- There are two queues now: ready queue and blocked queue
  - When the process is admitted in the system, it is placed in ready queue
  - When a process is removed from the processor, it is either placed in ready queue or in blocked queue (depending on circumstances)
  - When an event occurs, all the processes waiting on that event are moved from blocked queue onto ready queue.

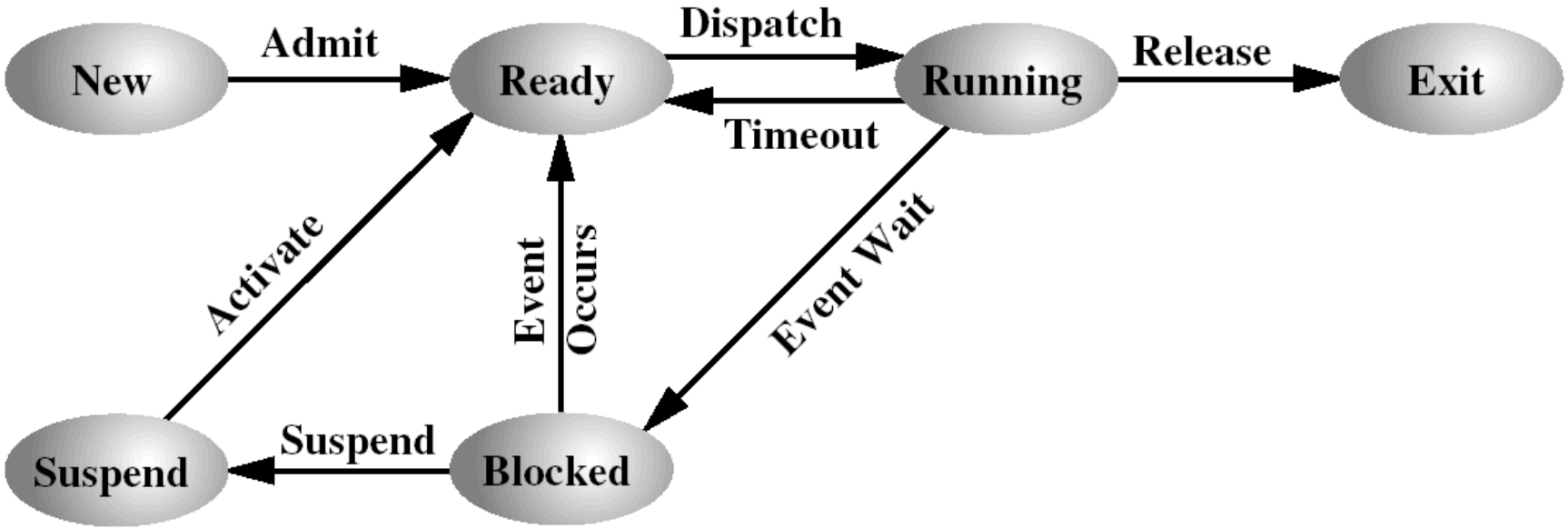


Queuing Discipline (2)



- Multiple blocked queues; one per each event
  - When event occurs, the entire list of processes is moved in ready queue

Suspended Processes



- 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 suspend state when swapped to disk

Process Management Services

- *create (&process_id, attributes)*
  - Creates a new process with implicit or specified attributes
- *delete (process_id)*
  - Sometime known as destroy, terminate or exit
  - Finishes the process specified by process_id
  - Whenever the process is terminated, all the files are closed,
  - all the allocated resources are released
- *abort (process_id)*
  - Same as the delete but for abnormal termination
  - Usually generates a :post mortem dump" which contains the state of the process before the abnormal termination
- *suspend (process_id)*
  - Determines the specified process to go in suspended state

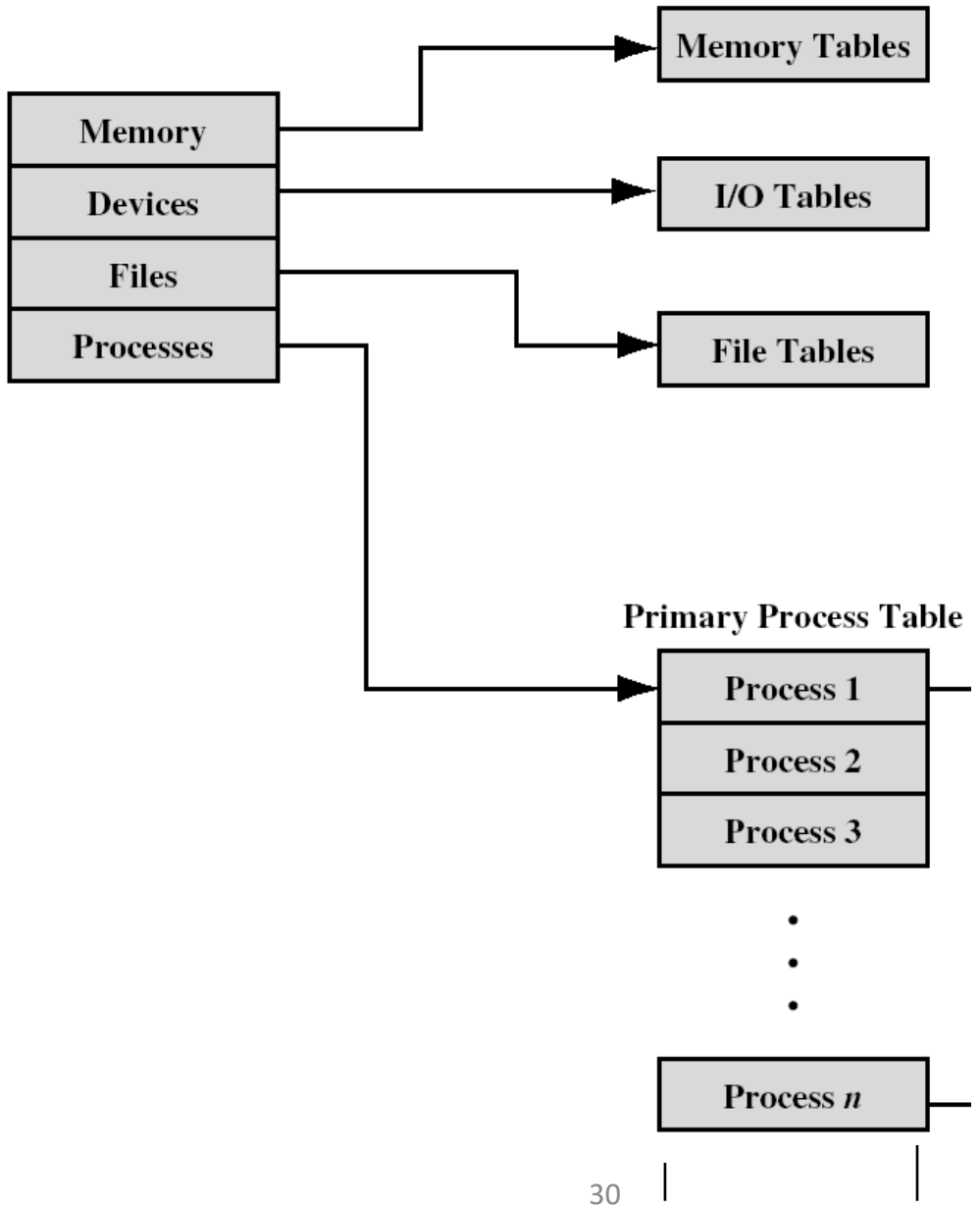
Process Management Services

- *resume (process_id)*
  - Determines the specified process to go from the suspended state in ready state
- *delay (process_id, time)*
  - Same with sleep
  - Suspends the specified process for the specified period of time
  - After the delay time elapses, the process is brought to ready state
- *get_attributes (process_id, &buffer_attributes)*
  - Used to find out the attributes for the given process
- *set_attributes (process_id, buffer_attributes)*
  - Used to set the attributes of the specified process

Process Description

Process Description

- What information does the operating system need to control processes and manage resources for them?
- Operating System Control Structures
  - Memory Tables
  - I/O Tables
  - File Tables
  - Primary Process Table
  - Process Image
    - User Program, user data, stack and attributes of the process



Memory Tables

- Used to keep track of both main (real) and secondary (virtual) memory.
  - Some of main memory is reserved for use by the operating system; the remainder is available to the processes.
- Contain:
  - The allocation of main memory to processes
  - The allocation of secondary memory to processes
  - Any *protection* attributes of blocks of main or virtual memory (such as which processes can access certain shared memory regions)
  - Any information needed to manage virtual memory

I/O Tables

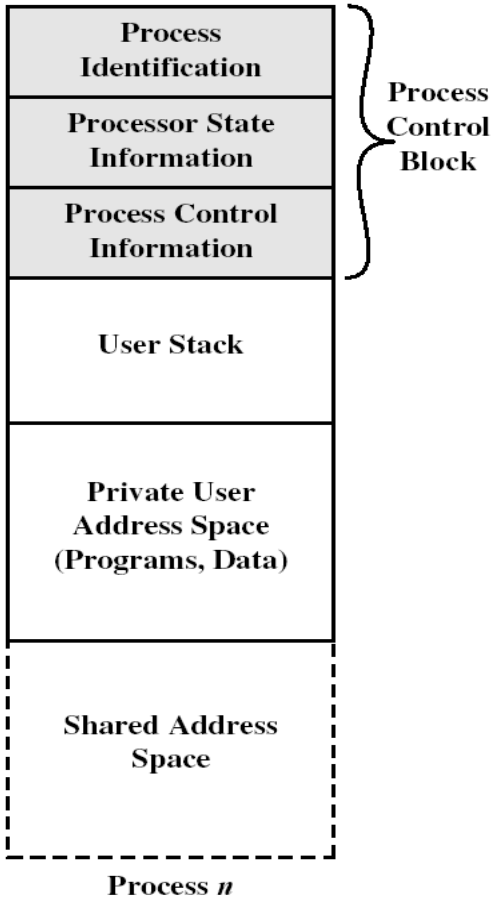
- Are used by the operating system to manage the I/O devices
  - At any given time, an I/O device may be available or assigned to a particular process.
  - If an I/O is in progress, the OS needs to know the status of the I/O operation and the location in main memory being used as the source or destination of the I/O transfer.

File Tables

- These tables provide information about:
  - the existence of files
  - their location on secondary memory
  - their current status
  - other attributes
- Much of this information is maintained and managed by the File Manager, in which case the process manager has little or no knowledge of files.

Process Tables

- Primary process table is used to keep one entry per each process in the operating system.
  - Each entry contains at least one pointer to a process image.
- The Process Image contains:
  - **Stack**
    - Each process has one or more stacks associated with it.
    - A stack is used to store parameters and calling addresses for procedure and system calls
  - **User Data**
    - Program data that can be modified, etc.
  - **Process Control Block**
    - Data needed by the operating system to control the process (attributes and information about process)



Process Control Block

Contains:

1. **Process Identification**: data always include a unique identifier for the process

2. **Processor State Information**: define the status of a process when it is suspended

3. **Process Control Information**: used by the OS to manage the process

Process Identification

- Identifiers
  - Numeric identifiers that may be stored with the Process Control Block include:
    - Identifier of this process
    - Identifier of the process that created this process (parent process)
    - User identifier

Processor State Information

- **User-Visible Registers**
  - A user-visible register is one that may be referenced by means of the machine language that the processor executes.
- **Control and Status Registers**
  - These are a variety of processor registers that are employed to control the operation of the processor. These include:
    - *Program Counter:* Contains the address of the next instruction to be fetched
    - *Condition codes:* Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow bits)
    - *Status information:* Includes interrupt enabled/disabled flags, execution mode
- **Stack Pointers**
  - Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls.
    - The stack pointer points to the top of the stack.

Process Control Information (1)

- **Scheduling and State Information**
- This is information that is needed by the operating system to perform its scheduling function. Typical items of information:
  - *Process state:* defines the readiness of the process to be scheduled for execution (e.g., running, ready, waiting, halted).
  - *Priority:* One or more fields may be used to describe the scheduling priority of the process. In some systems, several values are required (e.g., default, current, highest-allowable)
  - *Scheduling-related information:* This will depend on the scheduling algorithm used. Examples are the amount of time that the process has been waiting and the amount of time that the process executed the last time it was running.
  - *Event:* Identity of event the process is awaiting before it can be resumed

Process Control Information (2)

- **Data Structuring**
  - A process may be linked to another process in a queue or other structure. E.g.,:
    - all processes in a waiting state for a particular priority level may be linked in a queue.
    - a process may exhibit a parent-child (creator-created) relationship with another process. The process control block may contain pointers to other processes to support these structures.
- **Inter-process Communication**
  - Various flags, signals, and messages may be associated with communication between two independent processes.
- **Process Privileges**
  - Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed.
  - In addition, privileges may apply to the use of system utilities and services.

Process Control Information (3)

- **Memory Management**
  - This section may include pointers to segment and/or page tables that describe the virtual memory assigned to this process.

- **Resource Ownership and Utilization**
  - Resources controlled by the process may be indicated, such as opened files.
  - A history of utilisation of the processor or other resources may also be included
    - this information may be needed by the scheduler.

Threads and Processes

Threads and Processes

- A process is defined sometimes as a *heavyweight process*
- A thread is defined as a *lightweight process*

- Separate two ideas:
  - ***Process***: Ownership of memory, files, other resources

    -> execution of applications
  - ***Thread***: Unit of execution we use to dispatch

    -> share the same address space hence can read from and write to the same data structures

- **Multithreading**
  - Allow multiple threads per process

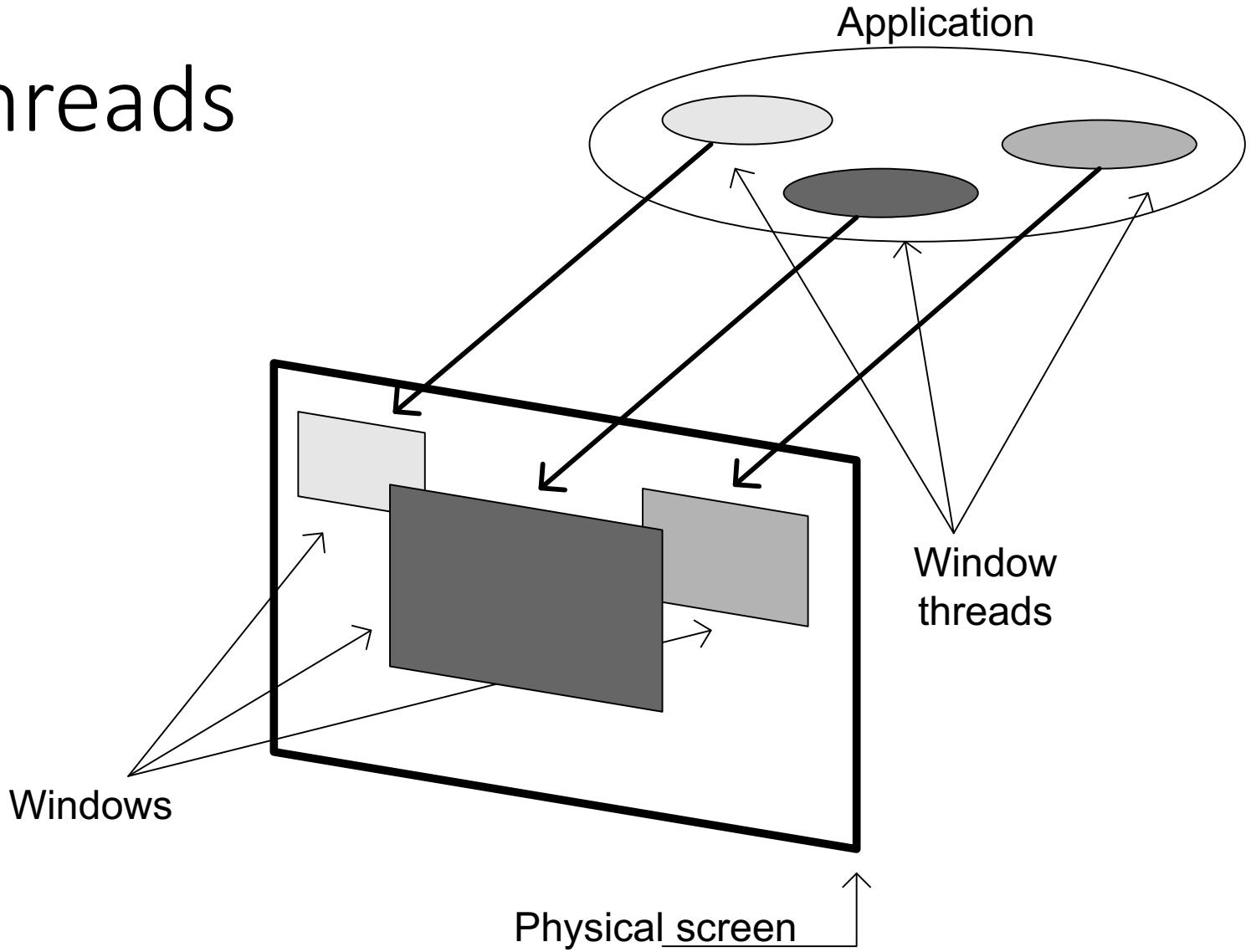
Threads (1)

- It is a unit of computation associated with a particular heavyweight process, using many of the associated process's resources
  - has a minimum of internal state and a minimum of allocated resources
- A group of threads are sharing the same resources:
  - E.g., files, memory space, etc.
- The process is the execution environment for a family of threads
  - a process with one thread is a *classic* process
- A thread belongs only to one process

Threads (2)

- Individual execution state
- Each thread has a control block, with a state (Running/Blocked/etc.), saved registers, instruction pointer
- Separate stack and hardware state (PC, registers, PSW, etc.) per thread
- Shares memory and files with other threads that are in that process
- Faster to create a thread than a process
- Because a family of threads belonging to the same process have common resources, the thread switch is very efficient
- Thread switch for threads from different processes is as complex as classic process switch

Using Threads



References

- “Operating Systems”, William Stallings, ISBN 0-13-032986-x
- “Operating Systems – A Modern Perspective”, Garry Nutt, ISBN 0-8053-1295-1