

# Processes

**Mohammad S. Hasan**

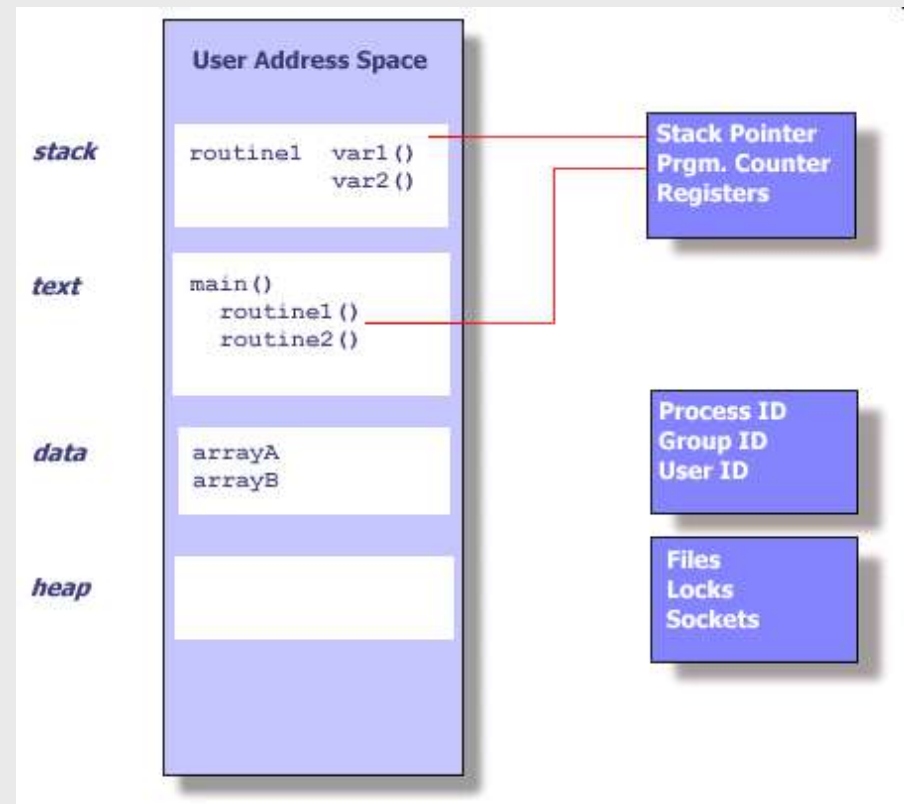
**Staffordshire University, UK**

# Lecture Outline

- ◆ **What is a process?**
- ◆ **Structure of a process**
- ◆ **Process state**
- ◆ **Resource queues**
- ◆ **Process scheduling**
- ◆ **Context switching**
- ◆ **Process creation and termination**
- ◆ **Co-operating processes**

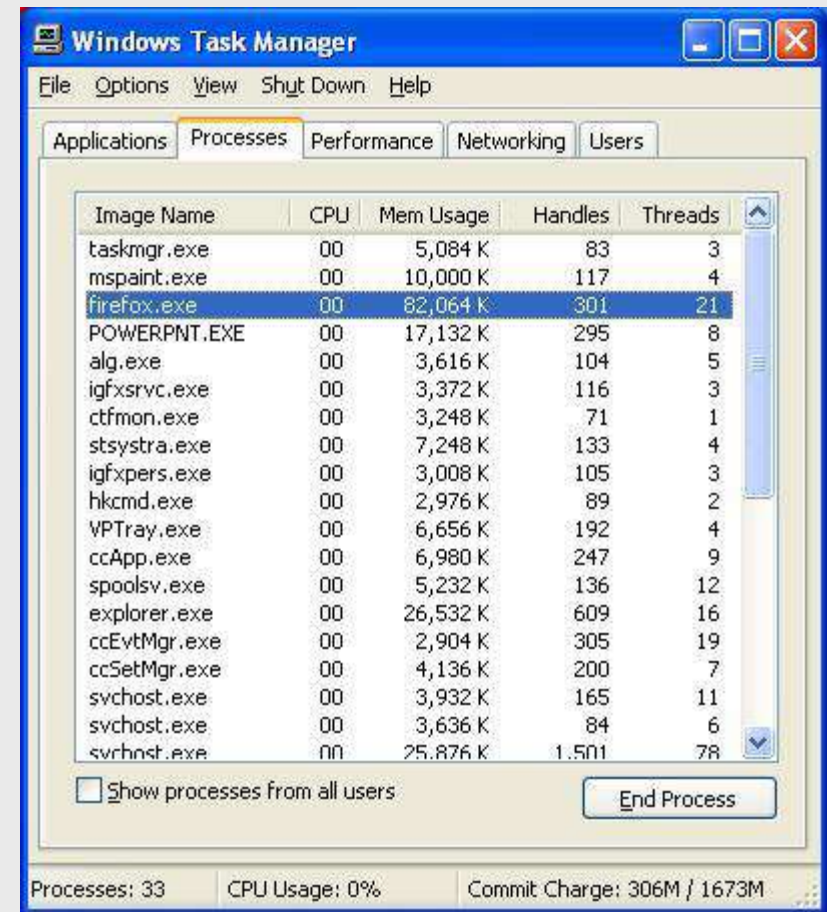
# Concept of a Process

- OS shares resources between user programs
- User program is simply a file of text that specifies what actions the computer should perform.
- A program becomes a process when it is loaded into memory for execution



# Concept of a Process (cont.)

- The executable version of the program is known as a process.
- It is the process that runs on the computer.
- Process is a program in execution.
- Processes are basic units that resources are shared between.



| Image Name   | CPU | Mem Usage | Handles | Threads |
|--------------|-----|-----------|---------|---------|
| taskmgr.exe  | 00  | 5,084 K   | 83      | 3       |
| mspaint.exe  | 00  | 10,000 K  | 117     | 4       |
| firefox.exe  | 00  | 82,064 K  | 301     | 21      |
| POWERPNT.EXE | 00  | 17,132 K  | 295     | 8       |
| alg.exe      | 00  | 3,616 K   | 104     | 5       |
| igfxsrv.exe  | 00  | 3,372 K   | 116     | 3       |
| ctfmon.exe   | 00  | 3,248 K   | 71      | 1       |
| stsysra.exe  | 00  | 7,248 K   | 133     | 4       |
| igfxpers.exe | 00  | 3,008 K   | 105     | 3       |
| hkcmd.exe    | 00  | 2,976 K   | 89      | 2       |
| VPTray.exe   | 00  | 6,656 K   | 192     | 4       |
| ccApp.exe    | 00  | 6,980 K   | 247     | 9       |
| spoolsv.exe  | 00  | 5,232 K   | 136     | 12      |
| explorer.exe | 00  | 26,532 K  | 609     | 16      |
| ccEvtMgr.exe | 00  | 2,904 K   | 305     | 19      |
| ccSetMgr.exe | 00  | 4,136 K   | 200     | 7       |
| svchost.exe  | 00  | 3,932 K   | 165     | 11      |
| svchost.exe  | 00  | 3,636 K   | 84      | 6       |
| svrhost.exe  | 00  | 25,876 K  | 1,501   | 78      |

Processes: 33    CPU Usage: 0%    Commit Charge: 306M / 1673M

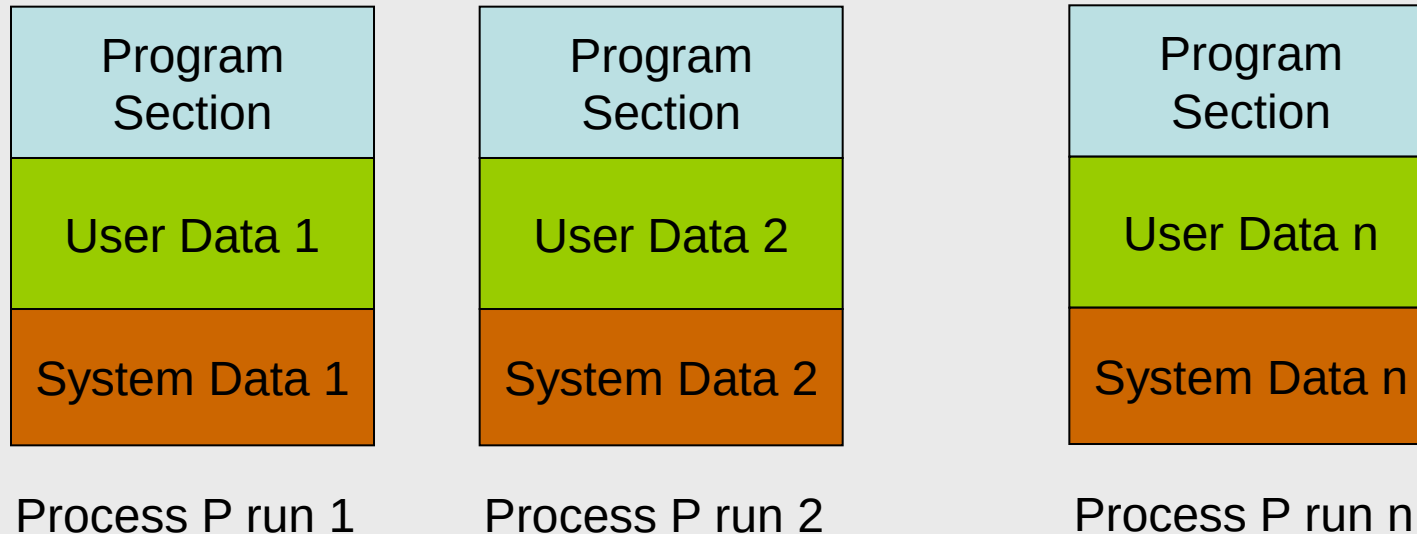
# Process Structure

## ◆ A process includes

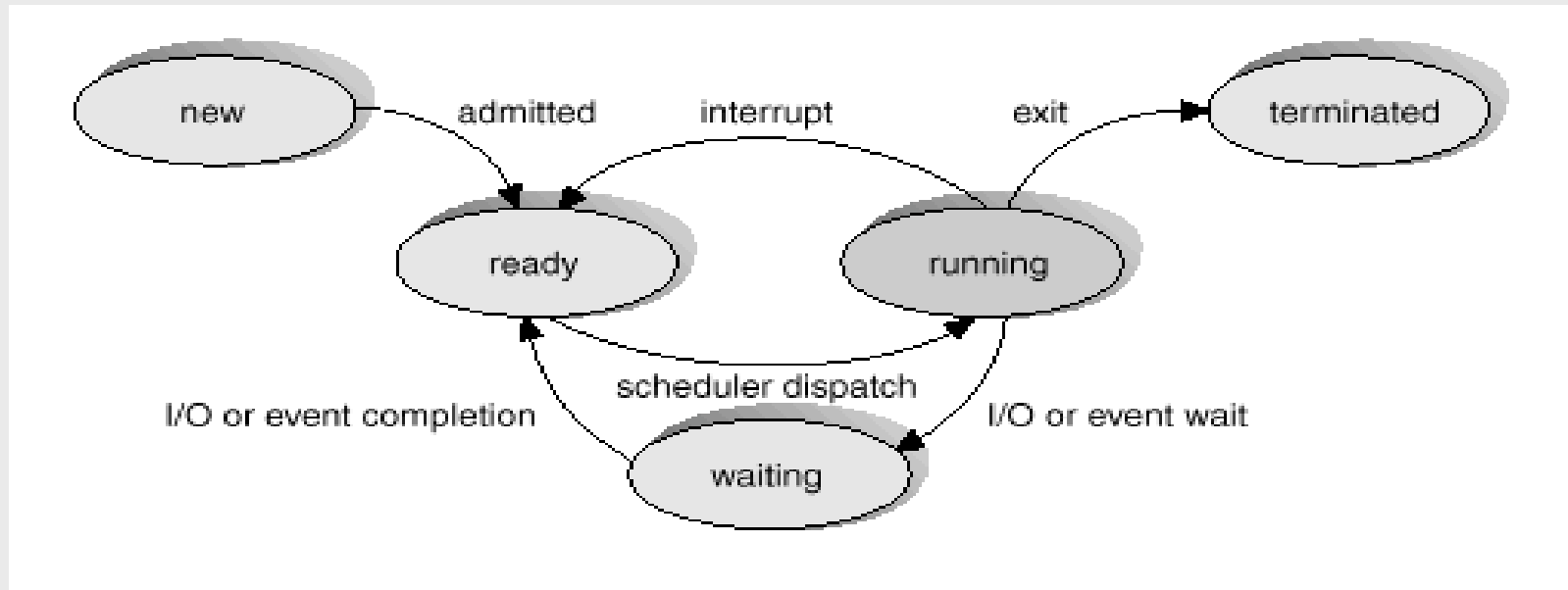
- **program section - contains a copy of the machine code instructions**
- **user data section - to hold variable data values**
- **system data section - area to hold**
  - **process context info for when process interrupted - program counter, processor registers, etc and**
  - **system information about resources allocated to process, etc.**

# Process Structure (cont.)

- user and system data sections obviously may be different from one run of a process to another



# Process State



# Process State (cont.)

➤ As a process executes, it changes state.

- **New**

- The process is created by OS

- **Ready**

- The process is waiting to be assigned to a processor.

- **Running**

- Instructions are being executed on processor

- **Waiting**

- The process is waiting for some event to occur

- **Terminated**

- The process has finished execution

# Process Control Block (PCB)

## ◆ Process Control Block (PCB)

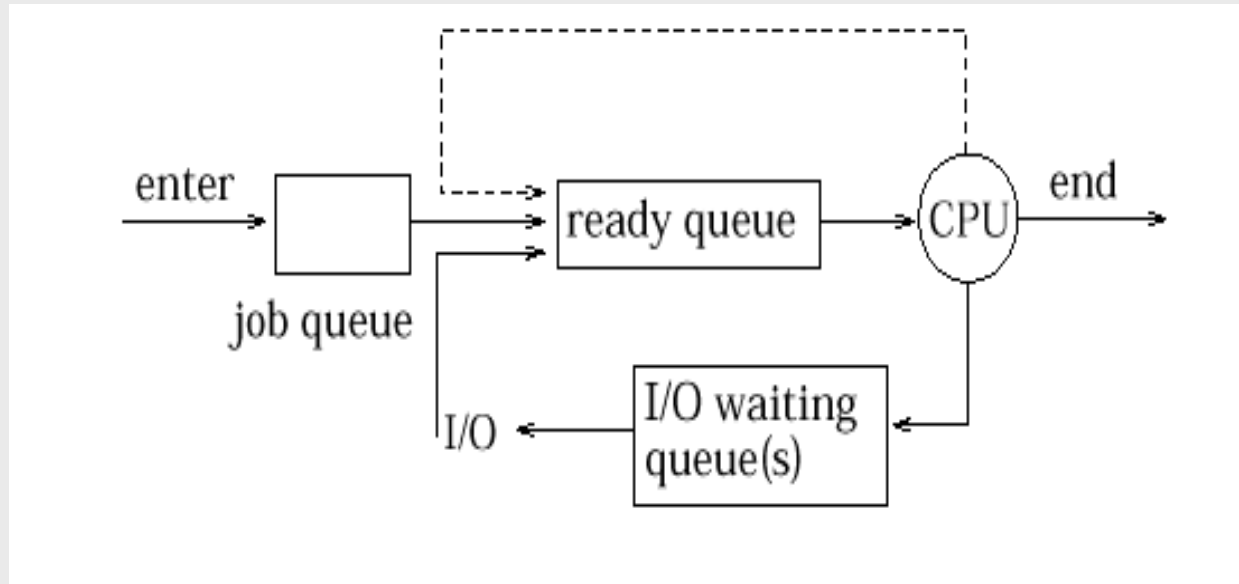
- data structure that contains Information associated with each process

|                    |               |
|--------------------|---------------|
| pointer            | process state |
| process number     |               |
| program counter    |               |
| registers          |               |
| memory limits      |               |
| list of open files |               |
| ⋮                  |               |

# Process Control Block (cont.)

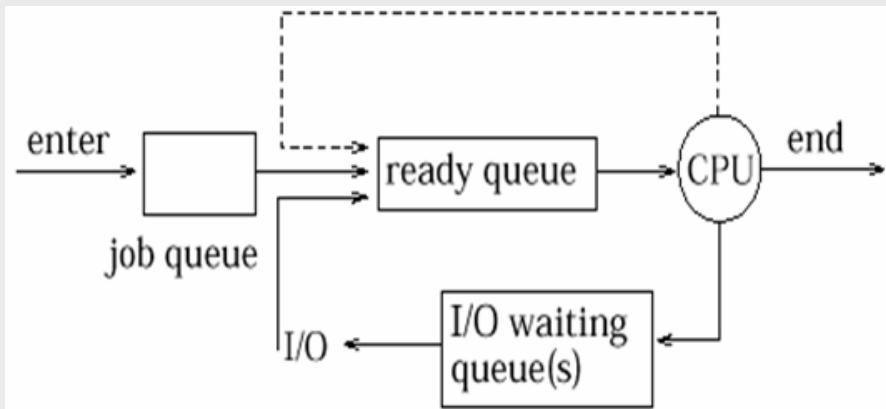
- ◆ **Process state and process ID number**
- ◆ **Copies of CPU registers when program not executing e.g. program counter, data, index, and other registers**
- ◆ **CPU scheduling information - e.g. priority level**
- ◆ **Memory-management information - e.g. base/limit register values**
- ◆ **Accounting information - CPU time used, etc**
- ◆ **I/O status information - which I/O resources allocated**

# Process Scheduling Queues



- Processes in ready/waiting states are held in queues
- Processes move between queues as they change state

# Process Scheduling Queues (cont.)



## Job queue

- set of all processes wanting to enter computer system

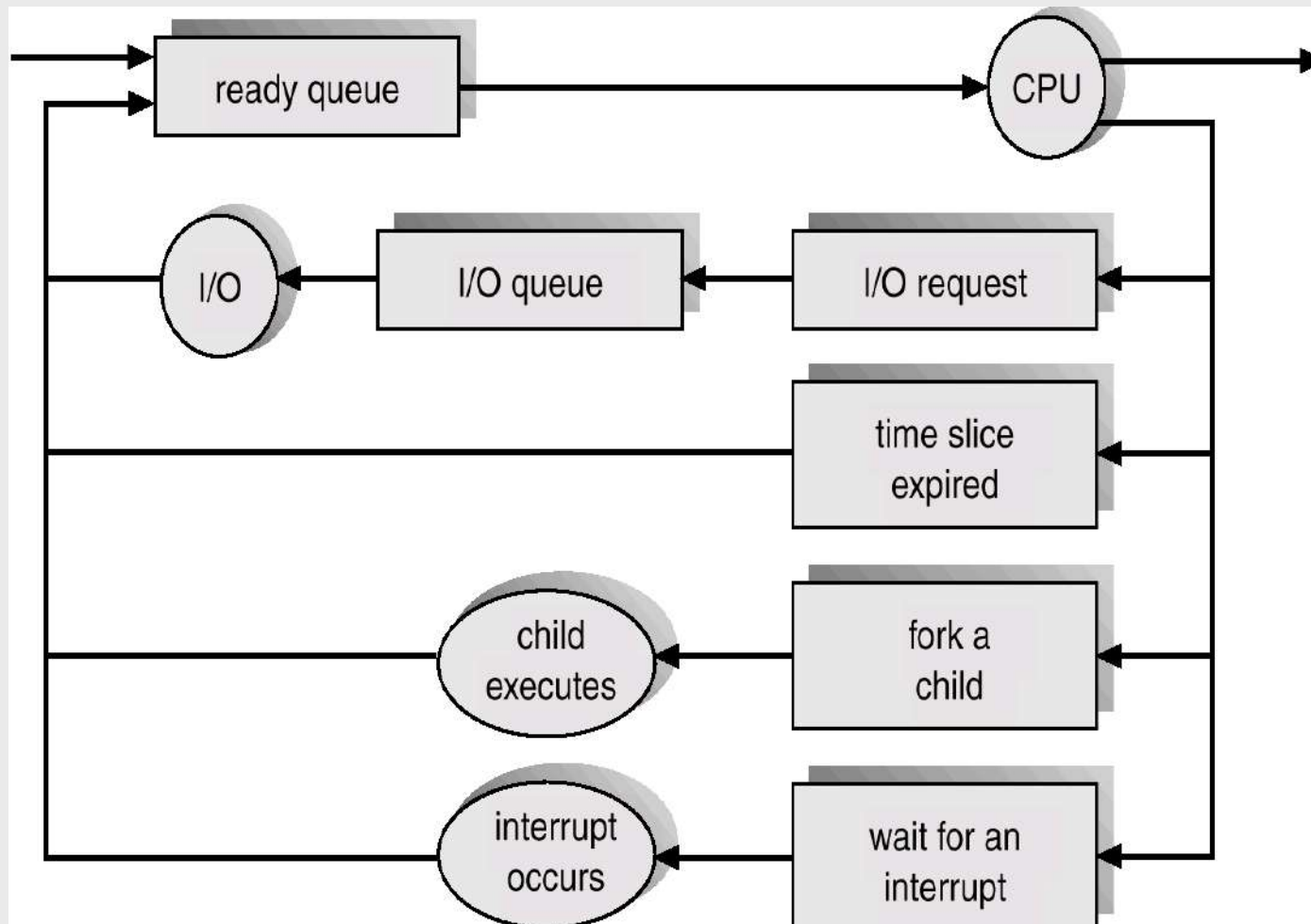
## Ready queue

- set of all processes that are in main memory, ready and waiting to execute - a linked list

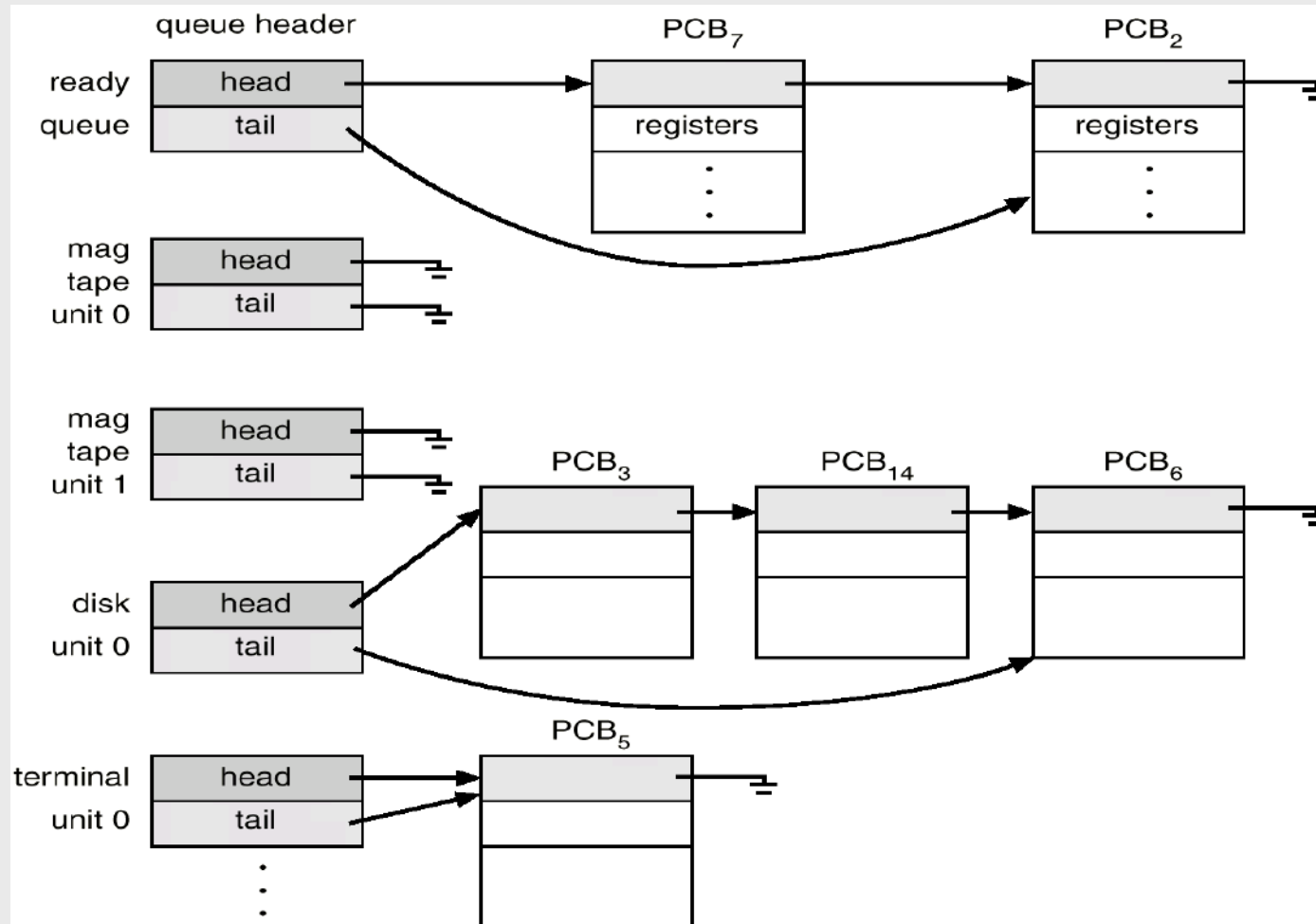
## Device queues

- set of processes waiting to use an I/O device

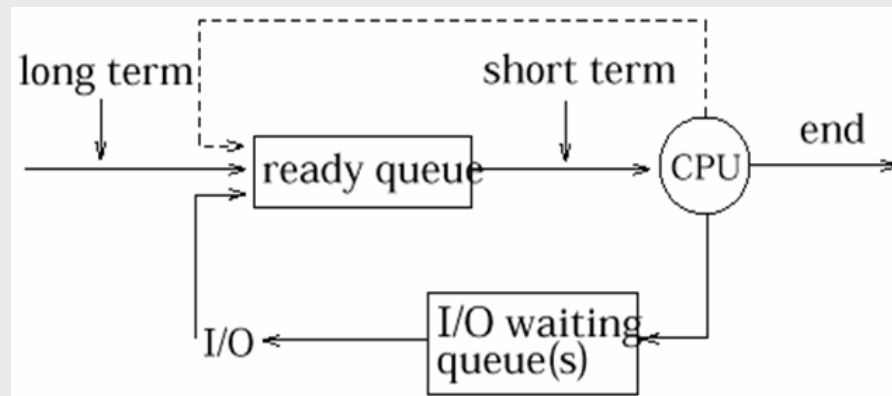
# Process queues and events



# Ready queue and device queues



# Execution Schedulers



- ◆ **Long-term scheduler (or job scheduler)**
  - selects programs should be brought into main memory for execution
- ◆ **Short-term scheduler (or CPU scheduler)**
  - selects which process should be executed next and allocates CPU to it.

# Execution Schedulers (cont.)

- ◆ **Short-term scheduler is invoked very frequently**
  - (milliseconds) -> (must be fast).
- ◆ **Long-term scheduler is invoked very infrequently**
  - (seconds, minutes) -> (may be slow).
- ◆ **The long-term scheduler controls the degree of multiprogramming**
  - the number of processes in the system that can be scheduled onto the CPU i.e. number of processes competing for CPU as a resource



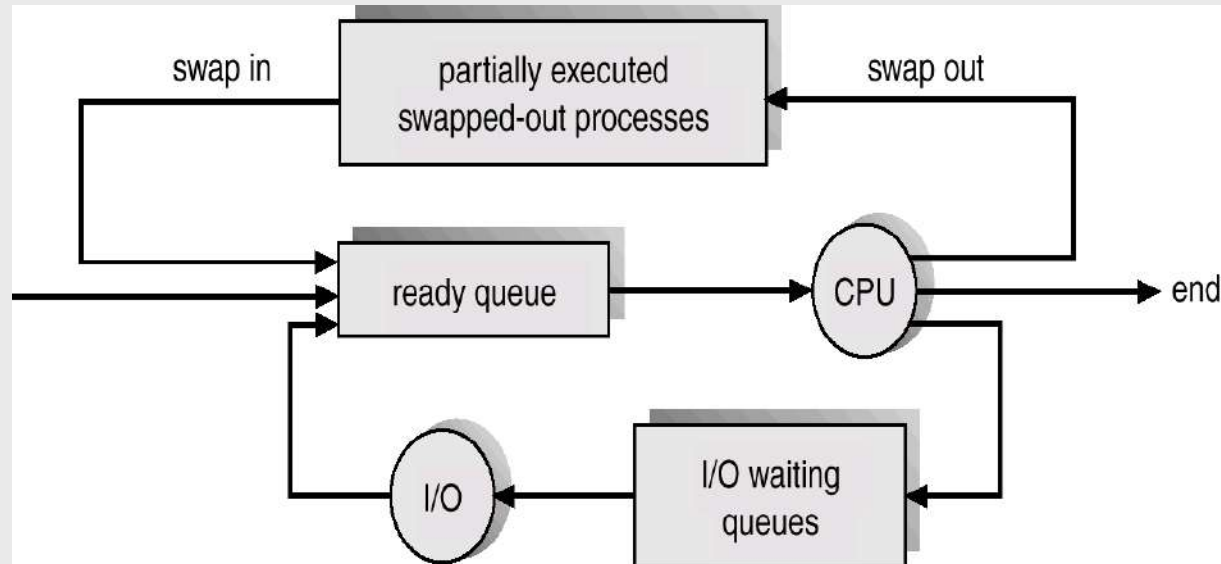
# Execution Schedulers (cont.)

## ◆ Process Types

- I/O-bound process – have small amount of computation before it needs to do some I/O; many I/O requests -Typical interactive user programs.
- CPU-bound process – large amount of computation before it needs to do some I/O; few I/O requests - Programs that require sustained periods of calculation e.g. modelling applications

## ◆ Long-term scheduler wants to balance use of CPU between I/O-bound and CPU-bound processes

# Medium Term Scheduler

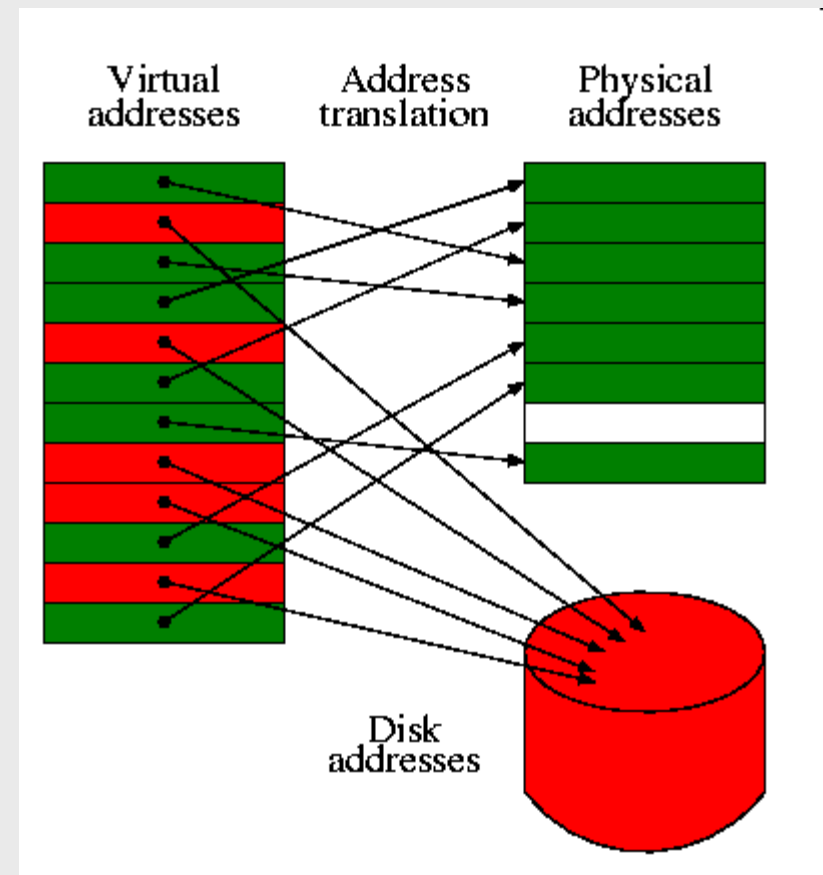


## ◆ Linked to Virtual memory

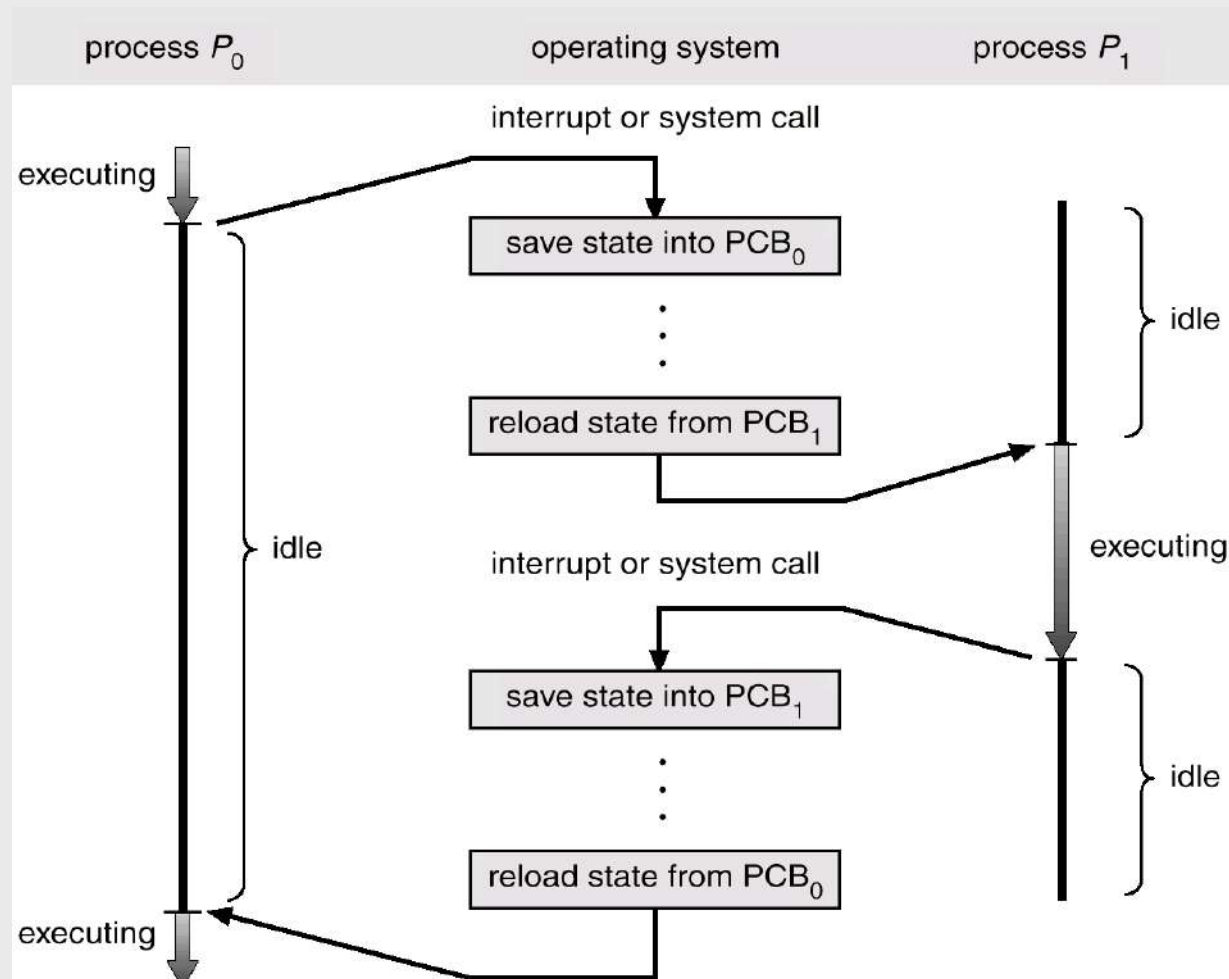
- Hard drive is used as the temporary storage of the running processes

# Medium Term Scheduler (cont.)

- **If number of executing processes is very high**
  - **System response will be slow**
  - **It can temporarily swap some processes out to disk and bring them back later.**
  - **It is intermediate between long and short term.**
- **May exist in time sharing systems to provide temporary relief from overloading**



# CPU switching between Processes



# Context Switch

- ◆ **When CPU switches to another process**
  - the system must save the state (registers, stack, and other temporary values) of the old process and
  - load the saved state for the new process.
- ◆ **Context-switch time is overhead**
  - the system does no useful work while switching.
- ◆ **Time taken dependent on hardware support.**

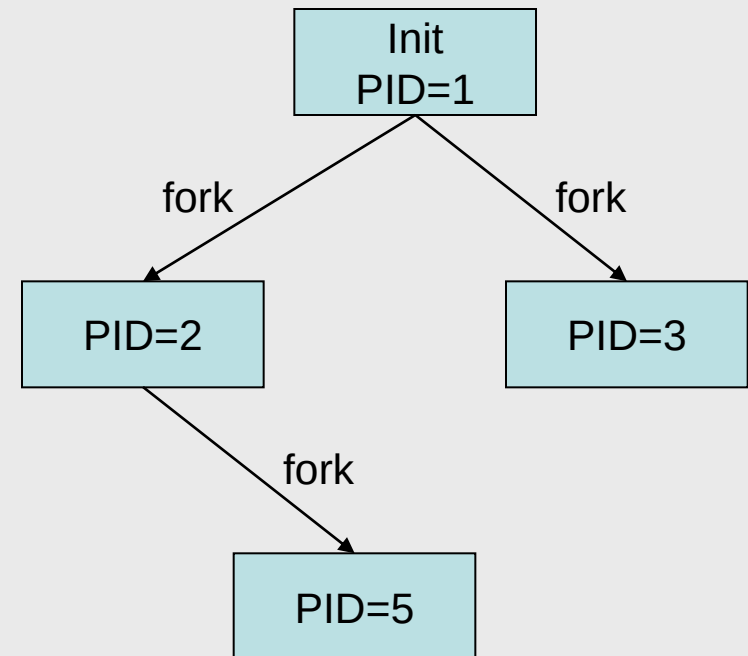
# Process Creation

## ◆ Process hierarchy

- A process can create a child process.
- The newly created child can create a child process of its own and so on.

## ◆ The first process or init process

- created at bootup of the OS and is part of OS.



# Process Creation (cont.)

## ◆ Resource sharing between parent and child processes

- Parent and children share all resources, or
- Children share subset of parent's resources, or
- Parent and child share no resources.

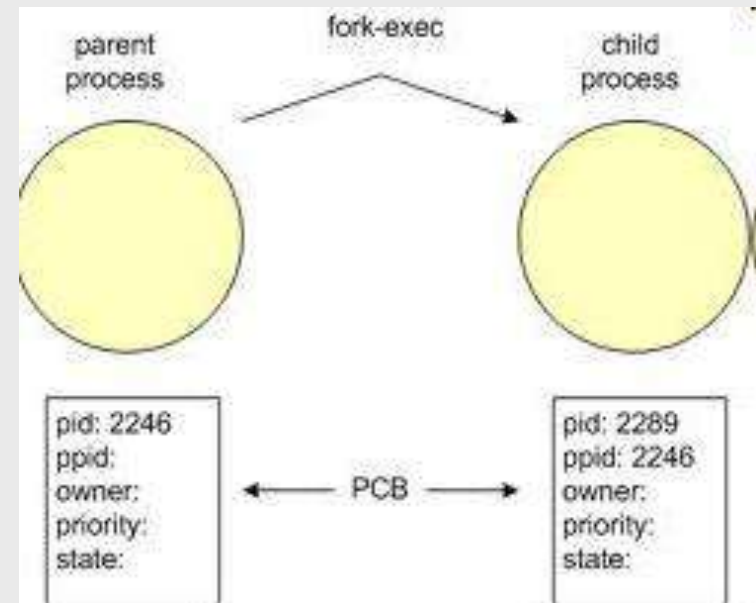
## ◆ Execution options

- Parent and children execute concurrently, or
- Parent waits until children terminate.

# Process Creation (cont.)

## UNIX example

- **fork system call creates new process which is a copy of the parent**
- **The exec system call**
  - **The exec system call can replace the child process's memory space with a different program**



# Process Termination

- ◆ **Process executes last statement and asks the operating system to delete it**
  - **exit system call**
- ◆ **Return data from child to parent (via wait).**
- ◆ **Process' resources are de-allocated by operating system.**

# Process Termination (cont.)

- ◆ **Parent may terminate execution of children processes (abort).**
- ◆ **Reasons for this could be**
  - **Child has exceeded allocated resources.**
  - **Task assigned to child is no longer required.**
  - **Parent is exiting and O/S does not allow children to continue if their parent terminates.**

# Cooperating Processes

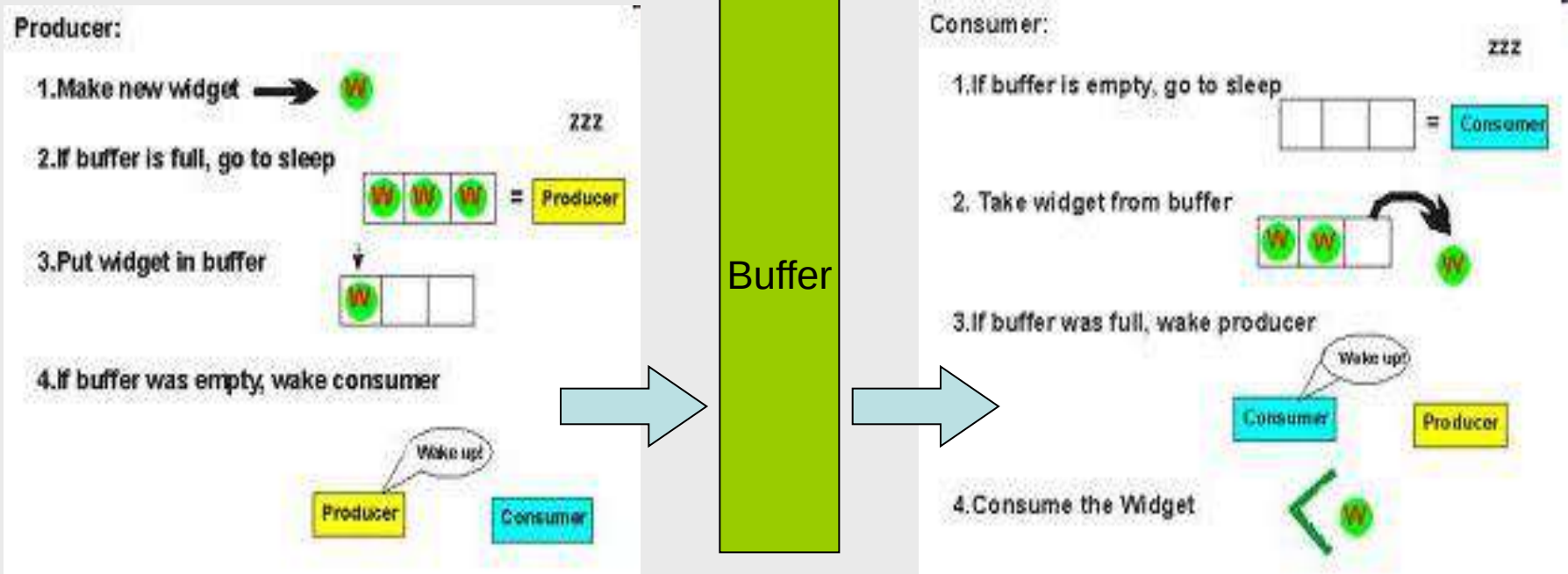
- ◆ **Process is independent if it cannot legally affect or be affected by the execution of another process.**
- ◆ **Process is cooperating if it can legally affect or be affected by the execution of another process.**

# Cooperating Processes (cont.)

## ◆ Advantages of process cooperation

- Information sharing
- Computation speed-up - but only if  $>1$  processor
- Modularity - dividing system functions into separate processes that then talk to each other
- way in which user process can interact with OS processes to do I/O, etc.

# Producer-Consumer Example



# Producer-Consumer Example (cont.)

## ◆ unbounded-buffer

- places no practical limit on the size of the buffer - consumer waits when buffer empty

## ◆ bounded-buffer

- assumes that there is a fixed buffer size - producer waits when buffer full and consumer when empty

# Bounded-Buffer – Shared-Memory solution

## ◆ Data structures

- **buffer for shared data - an array organised as a circular buffer**
- **in-index - where producer to put data**
- **out-index - where consumer to get data**

# Bounded-Buffer – Producer Process

## Repeat

- ....
- produce data item
- ....
- /\* buffer full when in-index is 1 less than out-index \*/
- while buffer full do
- no-op;
- buffer[in-index] = data item
- in-index = (in-index+1) modulo size-of-buffer

## until false

# Bounded-Buffer – Consumer Process

## Repeat

- `/* buffer empty when in-index == out-index */`
- `while buffer empty do`
- `no-op;`
- `get next data item from buffer[out-index]`
- `out-index = out-index+1 modulo size-of-buffer`
- `....`
- `consume data item`
- `....`

## until false

# References

## ◆ Operating System Concepts - Chapter 3.