OPERATING SYSTEMS

PROCESSES

OPERATING SYSTEM
Processes

What Is In This Chapter?

- Process Definition
- Scheduling Processes
- What Do Processes Do?
- Inter-process Communication

PROCESSES

Definitions

PROCESS CONCEPT:

A **program** is passive; a **process** active.
Attributes held by a process include
•hardware state,
•memory,
•CPU,
•progress (executing)

WHY HAVE PROCESSES?

Resource sharing (logical (files) and physical(hardware)).

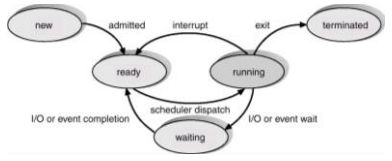
Computation speedup - taking advantage of multiprogramming – i.e. example of a customer/server database system.

Modularity for protection.

PROCESSES

PROCESS STATE

- **New** The process is just being put together.
- **Running** Instructions being executed. This running process holds the CPU.
- **Waiting** For an event (hardware, human, or another process.)
- **Ready** The process has all needed resources - waiting for CPU only.
- **Suspended** Another process has explicitly told this process to sleep. It will be awakened when a process explicitly awakens it.
- **Terminated** The process is being torn apart.



PROCESSES

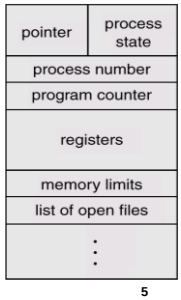
Process State

PROCESS CONTROL BLOCK:

CONTAINS INFORMATION ASSOCIATED WITH EACH PROCESS:

It's a data structure holding:

- PC, CPU registers,
- memory management information,
- accounting (time used, ID, ...)
- I/O status (such as file resources),
- scheduling data (relative priority, etc.)
- Process State (so running, suspended, etc. is simply a field in the PCB).



PROCESSES

Scheduling Components

The act of **Scheduling** a process means changing the active PCB pointed to by the CPU. Also called a **context switch.**

A context switch is essentially the same as a process switch - it means that the memory, as seen by one process is changed to the memory seen by another process.
See Figure 3 on Next Page

SCHEDULING QUEUES:

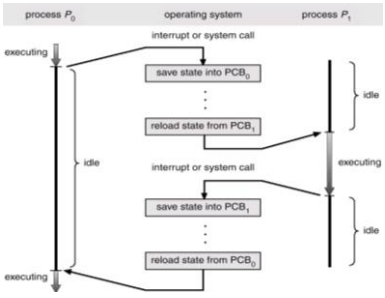
(Process is driven by events that are triggered by needs and availability)

- •Ready queue = contains those processes that are ready to run.
- •I/O queue (waiting state) = holds those processes waiting for I/O service.

What do the queues look like? They can be implemented as single or double linked. **See Figure Several Pages from Now**

PROCESSES

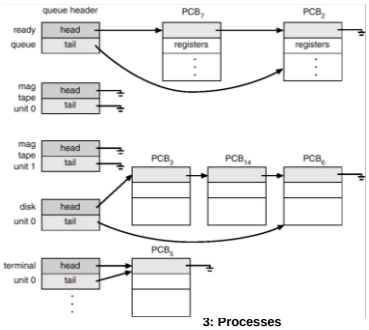
Scheduling Components



The CPU switching from one process to another.

PROCESSES

Scheduling Components



Ready Q And IO Q's

Figure 4.4

PROCESSES

Scheduling Components

LONG TERM SCHEDULER (job scheduler) : selects processes and load them into memory for execution

- Run seldom (when job comes into memory)
- Controls degree of multiprogramming
- Tries to balance arrival and departure rate through an appropriate job mix.

SHORT TERM SCHEDULER (CPU scheduler): selects processes ready for execution and allocate the CPU to one of them

Contains three functions:

- Code to remove a process from the processor at the end of its run.
  a)Process may go to ready queue or to a wait state.
- Code to put a process on the ready queue –
  a)Process must be ready to run.
  b)Process placed on queue based on priority.

PROCESSES

Scheduling Components

SHORT TERM SCHEDULER (cont.)

- Code to take a process off the ready queue and run that process (also called **dispatcher**).
  a) Always takes the first process on the queue (no intelligence required)
  b) Places the process on the processor.

This code runs frequently and so should be as short as possible.

MEDIUM TERM SCHEDULER

- Mixture of CPU and memory resource management.
- Swap out/in jobs to improve mix and to get memory.
- Controls change of priority.

PROCESSES

"fork" system call

- System calls: *fork* & *exec* are involved in process creation
- fork : a copy of the calling process is created. After execution of fork, there will be 2 identical processes running.

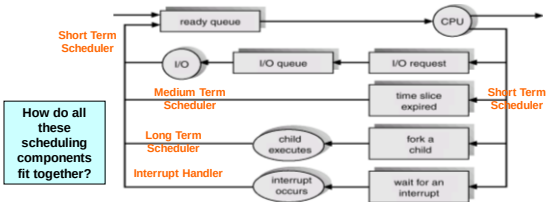
Initial process ID: 123
fork ----------→ child process ID: 391 –*exec*→ myprog & --→ child process ID:391
| execute myprog
|
parent process ID: 123
..continue

PROCESSES

Scheduling Components

INTERRUPT HANDLER

- In addition to doing device work, it also readies processes, moving them, for instance, from waiting to ready.



PROCESSES

Process Relationships

Parent can run concurrently with child, or wait for completion.

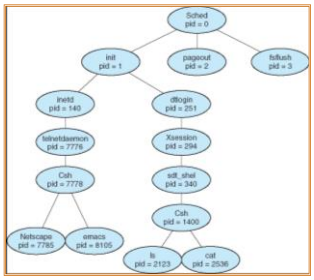
Child may share all (fork/join) or part of parent's variables.

Death of parent may force death of child.

Processes are static (never terminate) or dynamic (can terminate).

Independent Execution is deterministic and reproducible. Execution can be stopped/ started without affecting other processes.

Cooperating Execution depends on other processes or is time dependent. Here the same inputs won't always give the same outputs; the process depends on other external states.



PROCESSES

Interprocess Communication

Reasons for inter-process communication

1. Information sharing (concurrent access to shared files)
2. Computation speedup (complicated task broken up into several sub tasks, possible with multiple CPUs executing in parallel)
3. Modularity (modular design of systems)
4. Convenience (multitasking capabilities)

PROCESSES

Interprocess Communication

This is how processes talk to each other.

There are basically two methods:

Shared memory (with a process "kick") -- fast/ no data transfer.

Message Passing -- distributed/ better isolation.

FUNCTIONALITY OF COMMUNICATION LINKS:

- How are the links formed?
- How many processes on each link?
- How many links per pair of processes?
- Capacity - buffer space - can messages be enqueued.
- Message formats and sizes
- Uni- or bidirectional

METHODS OF IMPLEMENTATION:

- Direct or indirect - to process or mailbox.
- Symmetric or asymmetric?
- Buffering mechanism
- Send by copy or by reference?
- Fixed or variable size messages?

PROCESSES

Interprocess Communication

DIRECT COMMUNICATION:

Need to know name of sender/receiver. Mechanism looks like this:

send (Process_P, message) ;

receive (Process_Q , message);

receive (id, message) <-- from any sender

The Producer/Consumer Problem is a standard mechanism. One process produces items that are handed off to the consumer where they are "used".

```
repeat                          repeat
    produce item                    receive( producer,  nextp )
    send( consumer,  nextp)         consume item
until false                             until false
```


PROCESSES

Interprocess Communication

Other properties of Direct Communication:

- Link established automatically (when send or receive requested.)
- Only two processes in this form.
- One link per pair of processes.
- Generally Bi-directional
- Receiver may not need ID of sender.

Disadvantage of Direct Communication:

- The names of processes must be known - they can't be easily changed since they are explicitly named in the send and receive.

PROCESSES

Interprocess Communication

INDIRECT COMMUNICATION

- Processes communicate via a named mailbox rather than via a process name. Mechanism looks like this:

  open(mailbox_name);
  send (mailbox_name, message);
  receive (mailbox_name, message);

- Link is established if processes have a shared mailbox. So mailbox must be established before the send/receive.
- More than two processes are allowed to use the same mailbox.
- May cause confusion with multiple receivers - if several processes have outstanding receives on a mailbox, which one gets a message?

BUFFERING (whether direct or indirect, messages reside in a queue)

Options for such queue implementation:

- **Zero** capacity: sender must wait for recipient to get message.
- **Bounded** capacity: sender must wait for recipient if more than *n* messages in buffer. If not sender can continue executing some other process.
- **Unbounded** capacity: sender is never delayed. Infinite capacity in the queue.

Message Passing methods

- Local Procedure Calls (LPC) – in standalone machines

- Sockets

- Remote Procedure Calls (RPC)

– between networked computers

Local procedure call (LPC) : communication between processes in the same machine

-client opens a handle
-client sends a connection request
-server creates 2 com ports & returns the handle to one of them to the client
-client & server send messages via the corresponding port

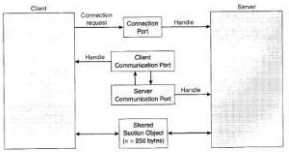


Figure 3.17 Local procedure calls in Windows XP.

Sockets – is identified by an IP address and a port number

Server "listens" to a client request through specific ports(http:80, ftp:21, smtp:25)

Useful in providing web, email, ftp services.

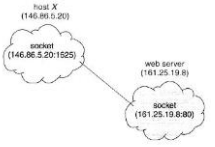


Figure 3.18 Communication using sockets.

Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.

Stubs – client-side proxy for the actual procedure on the server.

The client-side stub locates the server and *marshalls* the parameters.

The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server.

Useful in distributed file systems