PROCESSES

This lecture is about processes (threads of execution); what they are and how they cooperate.

PROCESS CONCEPT:

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

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.

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.

See the State diagram **<<< FIGURE 4.1 >>>**

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).

<<< FIGURE 4.2 >>>

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.

<<< FIGURE 4.3 >>>

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. **<<< FIGURE 4.4 >>>**

SCHEDULER COMPONENTS:

LONG TERM SCHEDULER

- 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

Contains two 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.
- Code to take a process off the ready queue and run that process (**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.

INTERRUPT HANDLER

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

How do all these scheduling pieces fit together?

See **<<< Figure 4.5 >>>**

OPERATIONS ON PROCESSES:

Creation	Concurrent vs. sequential parent and child. All vs. partial resource sharing (memory files, etc.) Resources given by parent to child.
Termination	Process completes code. Processes resources are exhausted. Process errors. Process resources must be cleaned up.
Suspension	Does the caller have the "privilege to do this? What is the current state of the target?
Change Priority	Does the caller have the "privilege to do this? What is the current state of the target?

PROCESS RELATIONSHIPS:

Shows parent/child relationships.

Parent can run concurrently with child or wait for completion.

Child may share all (fork/join) or part (as in UNIX) of parent’s variables.

Parent may kill child (when not needed anymore, or when child has exceeded its allocated resources.)

Death of parent usually forces death of child.

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

RELATION BETWEEN PROCESSES:

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.

THREADS:

Also called **light-weight processes**. Each thread contains its own PC, registers, stacks (State information).

Threads shared code, data, and files -- this entity is called a "**task**".

A task contains one or more threads. Threads execute sequentially and can be running/blocked/terminated.

Advantages:

Easy (and cheap) to schedule.
While one thread is blocked, others can run.
Any thread has access to other thread’s data.

Disadvantages:

No protection since all threads see all data in task.
Only one thread executing at any one instant.

Useful for servers <-- give an example of this.

Example of Threads in Solaris 2:

User level threads (support is in user level library rather than in kernel.)

These user level threads are associated with kernel based threads (Light weight process). User thread must have LWP in order to run.

Also kernel threads are used for operations not mapped to a user process.

<<< SEE FIGURE 4.9 >>>

INTERPROCESS COMMUNICATIONS:

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?

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
```

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.

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.

- A process can open many mailboxes.

- Used for both Uni- or bidirectional communication.

- May cause confusion with multiple receivers - if several processes have outstanding receives on a mailbox, which one gets a message?

- Solutions:

  Let link only have 2 processes. Let only one process do the receive. Let system choose, but do NOT let both processes get message

- Mailboxes are User or System created. User ownership avoids confusion. When one process dies, how does the twin process "know"? If done by system, then must provide create/send/receive/destroy.

BUFFERING:

Options include:

- **Zero** -- sender must wait for recipient to get message. Provides a rendezvous.
- **Bounded** -- sender must wait for recipient if more than n messages in buffer.
- **Unbounded** -- sender is never delayed.

MESSAGE FORMAT:

- Fixed, Variable, or Typed (as in language typing) size messages.
- Send reference rather than copy (good for large messages).
- Suspended vs. unsuspended sends.

GIVEN AN EXCEPTION:

- Process terminates (sender or receiver); system may terminate or notify other mailbox partner.
- A lost message can have many options:
- Left up to sender.
- OS detects, resends itself.
- OS detects, tells sender.