Module 4: Processes

- Process Concept
- Process Scheduling
- Operation on Processes
- Cooperating Processes
- Interprocess Communication

Definitions

- ***Algorithm:*** a logical procedure that in a finite number of steps solves a problem
- ***Program:*** formal expression of an algorithm by means od a programming language

A programming language needs a *run-time support*

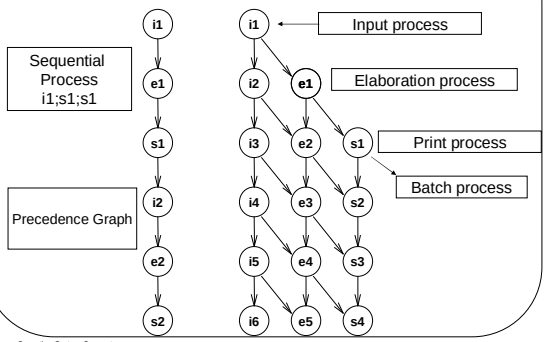
Process Concept

- ***Process:*** a **sequence** of operations performed by a program in execution on a given set of input data.
- The temporal behavior of a process can be analyzed through its *trace*
- ***Sequential process:*** is a deterministic process
  - Given a set of input data always produces the same output independently from
  - the start executions time
  - the execution speed
  - the number of active processes on the same system

Process Concept

- Process – a sequence of operations performed by a program in execution on a given set of input data.
- The temporal behavior of a process can be analyzed through its *trace*
- A process includes:
  - program counter
  - stack
  - data section

Sequential and Concurrent Processes



Process State

- As a process executes, it changes *state*
  - submit: The process is being created.
  - hold: The process wait for main memory resource.
  - running: Instructions are being executed.
  - waiting: The process is waiting for some event to occur.
  - ready: The process is waiting to be assigned to a processor.
  - complete: The process has finished execution.

Process State Diagram

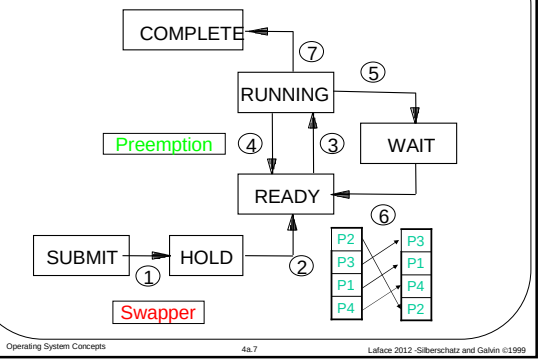
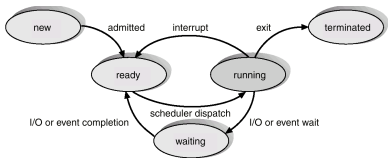


Diagram of Process State

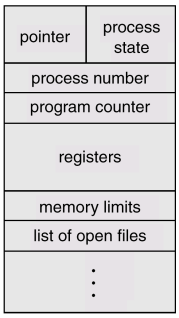


Process Control Block (PCB)

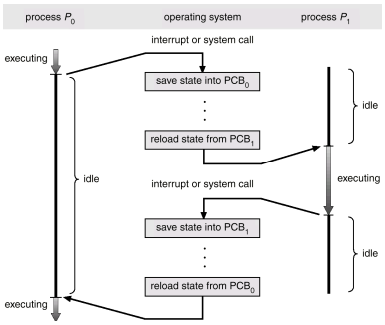
Information associated with each process.

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- I/O status information
- Accounting information

Process Control Block (PCB)



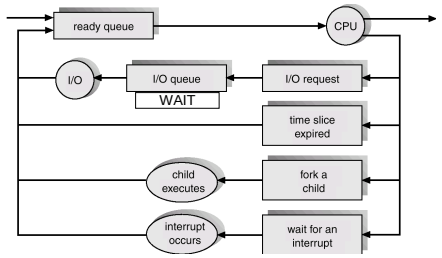
CPU Switch From Process to Process



Context Switching

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- Time devoted to context switching is *overhead, i.e.* work not directly useful for any process.
- Context switching time must be minimum. It is dependent on HW support.

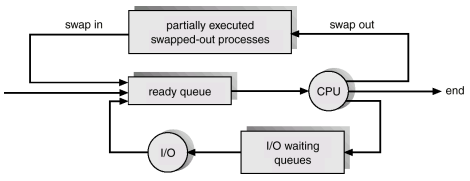
Representation of Process Scheduling



Schedulers

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.
- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU.

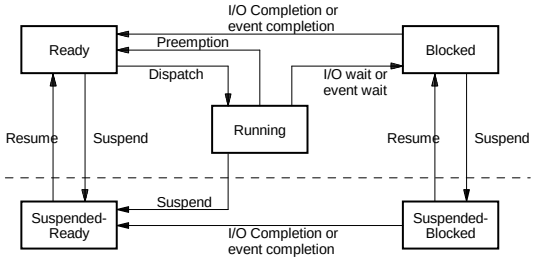
Addition of Medium Term Scheduling



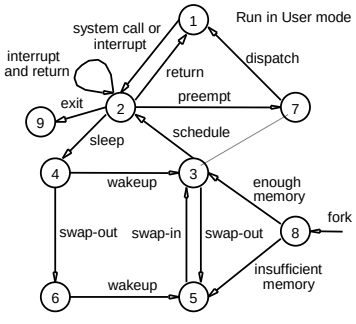
Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast).
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow).
- The medium-term scheduler controls the *degree of multiprogramming.*
- Processes can be described as either:
  - I/O-*bound process* – spends more time doing I/O than computations, many short CPU bursts.
  - CPU-*bound process* – spends more time doing computations; few very long CPU bursts.

Process State Diagram (more detailed)



Process State Diagram - UNIX



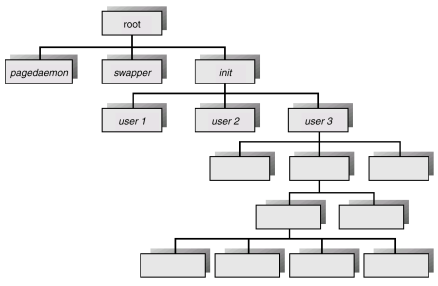
Process Creation

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent's resources.
  - Parent and child share no resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

Process Creation (Cont.)

- Address space
  - Child duplicate of parent.
  - Child has a program loaded into it.
- UNIX examples
  - **fork** system call creates new process
  - **execve** system call used after a **fork** to replace the process' memory space with a new program.

A Tree of Processes On a UNIX System



fork

fork0.c

alarm.c Sequential Alarm Clock

alarm-fork.c Concurrent Alarm Clock

fork3.c Erroneous generation of K childrens

fork-K-children.c

Process Address Space

fork1.c Separate address space

Process Termination

- Process terminates if it executes the system call **exit** or the last statement.
- The child can pass a byte to the parent (via **wait**).
- The parent receive the information by means of the **wait** argument.
- The OS kernel recovers the resources used by the child process (via **wait**).

Fork, wait and exit

fork2.c Fork wait and exit

wait.c

fork - exec

execlp.c

execve.c

UNIX Shell Skeleton

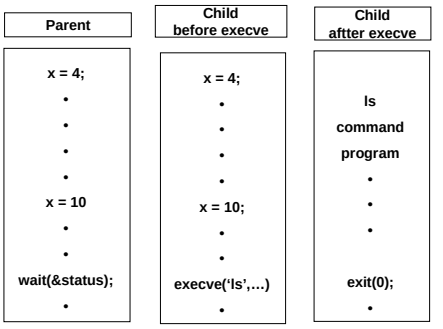
```
while ( TRUE ){
  write_prompt;
  read_command( command, parameters );
  if ( fork() != 0 )
                /* parent process */
    wait ( &status ); /* wait */
  else
                /* child process */
    execve( command, parameters );
}
```

UNIX Shell Skeleton

Background command
`ls –R / > /dev/null &`

```
while ( TRUE ){
  write_prompt;
  read_command( command, parameters );
  if ( fork() != 0 )
              /* parent process */
   /* wait ( &status ); don't wait*/
  else
              /* child process */
    execve( command, parameters );
}
```


Process Address Space



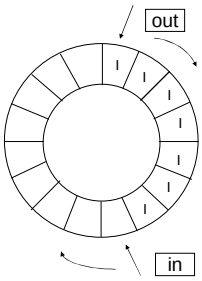
Cooperating Processes

- *Independent* process cannot affect or be affected by the execution of another process.
- *Cooperating* process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process.
  - *unbounded-buffer* places no practical limit on the size of the buffer.
  - *bounded-buffer* assumes that there is a limited buffer size.

Circular Buffer



Bounded-Buffer – Shared-Memory Solution

```
#define N 10
 typedef ….. Item ;  Item buffer[N];
 in = 0, out=0;
```

- Producer process

```
while (true) {
     …
     / * produces an item in nextp */
     …
     while ((in+1) % N == out) ;
     buffer [in] = nextp;
     in = (in+1) % N;
}
```

!!!!

Bounded-Buffer (Cont.)

- Consumer Process

```
while (true) {
    while (in == out ) ;
    nextc = buffer [out];
    out = (out+1) % N;
      …
    / * consumes the item in nextc */
      …
    }
```

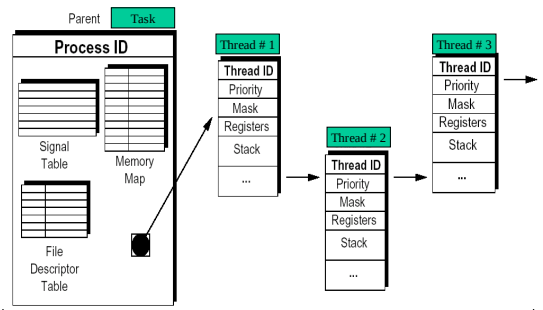
- The solution is correct for N != 1.
- But it wastes one buffer element.

Threads

- A *thread* (or *lightweight process*) is a basic unit of CPU utilization; it consists of:
  - program counter
  - register set
  - stack space
- A thread shares with its peer threads its:
  - code section
  - data section
  - operating-system resources

  collectively know as a the ***task*** *domain*.
- A traditional or *heavyweight* process is equal to a task with one thread

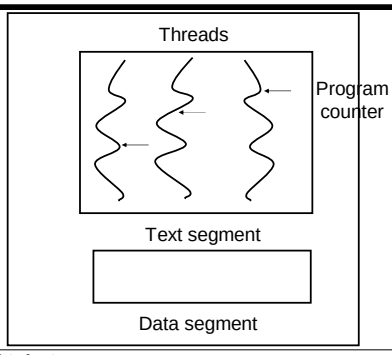
Threads



Threads (Cont.)

- In a multiple threaded task, while one server thread is blocked and waiting, a second thread in the same task can run.
- Cooperation of multiple threads in same job confers higher throughput and improved performance.
- Threads allow
  - concurrent execution of parts of the same application
  - service of several clients using copies of a code section
  - sequential processes to make blocking system calls while also achieving parallelism
  - exploiting multiprocessor boards

Multiple Threads within a Task



Threads (Cont.)

- Kernel-supported threads (Mach and OS/2).
- User-level threads; supported above the kernel, via a set of library calls at the user level (Project Andrew from CMU).
- Hybrid approach implements both user-level and kernel-supported threads (Solaris 2).

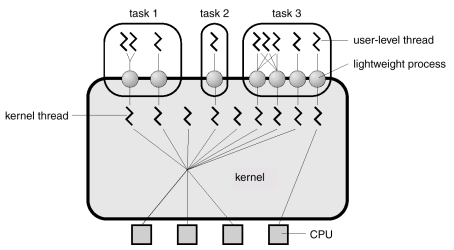
User threads

- Pros
  - Fast context switching between threads of the same task.
  - Can been implemented on top of any kernel
- Cons
  - If a thread performs a blocking system call, all the threads of the same task are blocked
  - A single thread per task running even on a multiprocessor system

Kernel threads

- Pros
  - The ready threads can be scheduled even if they belong to the task of a thread that has issued a blocking system call
  - Multiple thread per task can be executed on a multiprocessor system
- Cons
  - More expensive context switching because it needs the passage to kernel mode and kernel support
  - Constraints on the maximum number of thread per task/system

Hybrid approach - Solaris 2 Threads



Pthreads Examples

self.c

creation-and-join.c

try-exit.c

try-common-memory