Procesos

Agenda

1. Objetivos
2. Concepto de proceso
3. Planificación de procesos
4. Operaciones sobre los procesos
5. Comunicación Interprocesos
6. Comunicación en los sistemas Cliente-Servidor

UNIVERSIDAD DEL BÍO-BÍO

Objetivos

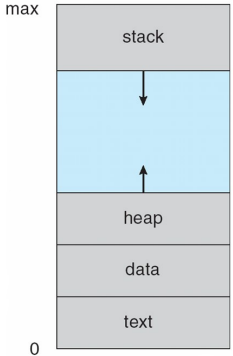
- Presentar el concepto de proceso (un programa en ejecución), en el que se basa todo el funcionamiento de un sistema informático.
- Describir los diversos mecanismos relacionados con los procesos, incluyendo los de planificación, creación y finalización de procesos, y los mecanismos de comunicación.

Concepto de proceso

- Un Sistema Operativo ejecuta una variedad de programas:
  - Procesadores de texto, navegadores, clientes de correo, juegos, base de datos, etc.
- Procesos: Un programa en ejecución.
- Un proceso incluye además del código:
  - contador de programa (actividad actual)
  - pila - stack (contiene datos temporales: parámetros de funciones, variables locales, etc)
  - sección de datos (variables globales)

Estructura de un proceso en Memoria

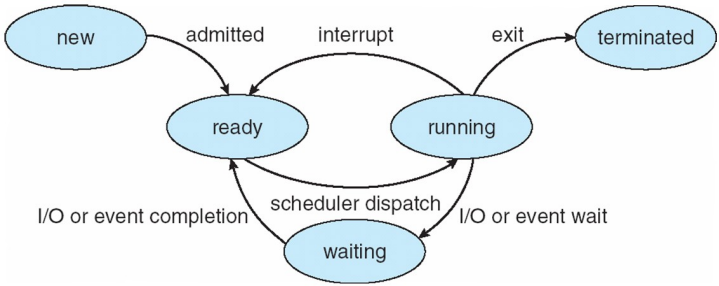
- Text - Código del programa.
- Heap - Memoria global y reservada.



Estado del proceso

- Nuevo: El proceso está siendo creado.
- En ejecución: Se están ejecutando las instrucciones.
- En espera: El proceso está esperando a que se produzca un suceso (como la terminación de una operación de E/S o la recepción de una señal).
- Preparado (listo): El proceso está a la espera de que le asignen a un procesador.
- Terminado: Ha terminado la ejecución del proceso.

Estado del proceso

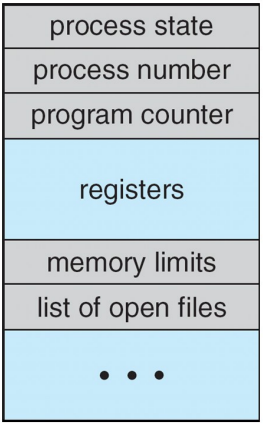


Bloque de control de proceso

- Cada proceso se representa en el SO mediante un Bloque de control de proceso **(PCB, process control block).**
- Contiene información asociada a un proceso específico:
  - Estado del proceso: Puede ser nuevo, preparado, en ejecución, en espera, detenido, etc.
  - Contador de programa: indica la dirección de la siguiente instrucción que va a ejecutar dicho proceso.
  - Registros de la CPU: acumuladores, punteros de pila, etc.
  - Información de planificación de la CPU: prioridad del proceso, punteros a las colas de planificación.
  - Información de gestión de memoria: valores registros base y límite, tablas de páginas o de segmentos, etc.
  - Información contable: cantidad de CPU y tiempo real empleados, número del proceso, etc.
  - Información de estado de E/S: lista de dispositivos de E/S asignados al proceso, archivos abiertos, etc.

UNIVERSIDAD DEL BÍO-BÍO

Bloque de control de proceso



Planificación de procesos

- Objetivo de **multiprogramación** es tener en ejecución varios procesos al mismo tiempo **para maximizar el uso de CPU.**
- Objetivo de **tiempo compartido** es **conmutar la CPU** entre distintos procesos **con tanta frecuencia** para que usuarios interactuen con cada programa mientras éste se ejecuta.
- Para cumplir estos objetivos el Planificación de procesos selecciona un proceso disponible para ejecutar el programa en la CPU.

Colas de planificación

- A medida que procesos entran al sistema se colocan en Cola de Trabajos (contiene todos los procesos del sistema).
- **Cola de procesos Preparados (listos)**. Residentes en MP, listos y esperando para ser ejecutados.
- **Cola de espera**. Procesos esperando por alguna operación de E/S.
- Colas se almacenan en forma de listas enlazadas. Cabeza de cola apunta al primer y último PCB.

Cola de procesos listos y de espera

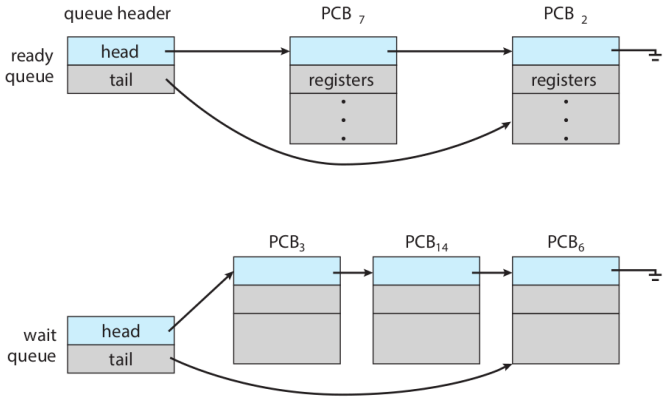
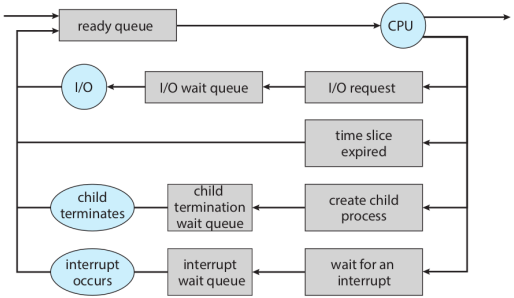


Diagrama de colas

- Representación que explica la planificación de procesos.
- Dos tipos de colas son presentados:
  - Cola de procesos listos.
  - Conjunto de colas de espera.
- Círculos representan los recursos que dan servicio a las colas.
- Flechas indican flujo del proceso en el sistema.

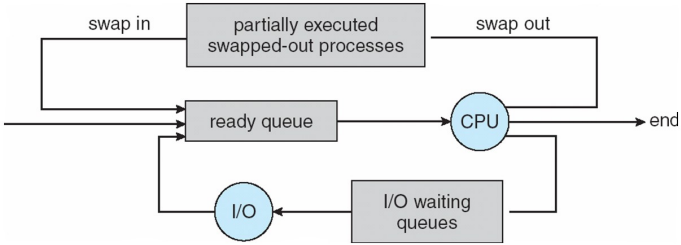


Planificación de CPU

- Procesos se mueven entre diversas colas durante su ciclo de vida.
- Planificador de CPU selecciona procesos desde cola de procesos listos y asigna CPU.
- Selección de proceso es muy frecuente (Normalmente, una vez cada 100 milisegundos o menos).

Planificación intermedia - swapping

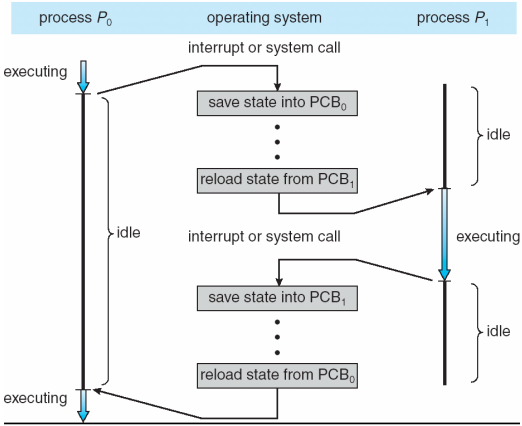
- Algunos sistemas operativos tienen este nivel intermedio de planificación.
- Para reducir grado de multiprogramación, mejorar mezcla de procesos, para liberar memoria, etc.
- Intercambio, planificador descarga (swap out) y vuelve a cargar el proceso (swap in) a memoria.



Cambio de contexto

- Interrupciones hacen que el SO oblique a la CPU abandonar su tarea actual para ejecutar una rutina del kernel.
- Cuando la CPU cambia a otro proceso, el sistema debe guardar el estado del proceso **viejo** y cargar el estado guardado para el nuevo proceso mediante un cambio de contexto.
- Contexto se almacena en el PCB del proceso.
- Tiempo utilizado en el cambio de contexto es desperdiciado, no se realiza trabajo útil durante conmutación.

Cambio de contexto



Operaciones sobre los procesos

- En la mayoría de los sistemas, procesos pueden ejecutarse concurrentemente y pueden crearse y eliminarse dinámicamente.
- Deben existir mecanismos para la creación y eliminación de procesos.

Creación de procesos

- Un proceso puede crear varios procesos nuevos mientras se ejecuta (uso de llamadas al sistema).
- Proceso creador = Padre, Nuevos procesos = Hijos de dicho proceso.
- Cada proceso Hijo puede crear otros procesos (árbol de procesos).
- Identificador de proceso unívoco (pid, process identifier).
- Proceso necesita de recursos. Cuando crea subproceso, dicho subproceso:
  - puede obtener recursos propios.
  - puede obtener subconjunto de recursos del proceso padre.

Creación de procesos

- Cuando un proceso crea otro proceso nuevo, dos posibilidades de ejecución:
  - El padre continúa ejecutandose concurrentemente con su hijo.
  - El padre espera hasta que alguno o todos sus hijos hayan terminado de ejecutarse. Llamada a sistema **wait()**
- Dos posibilidades en función del espacio de direcciones del nuevo proceso:
  - Hijo es un duplicado del Padre (usa el mismo programa y mismos datos del padre).
  - Hijo carga un nuevo programa.
- Ejemplo en UNIX:
  - Llamada a sistema **fork()** crea un nuevo proceso.
  - Llamada a sistema **exec()** usada después de fork para reemplazar el espacio de memoria del proceso con un nuevo programa.

UNIVERSIDAD DEL BÍO-BÍO

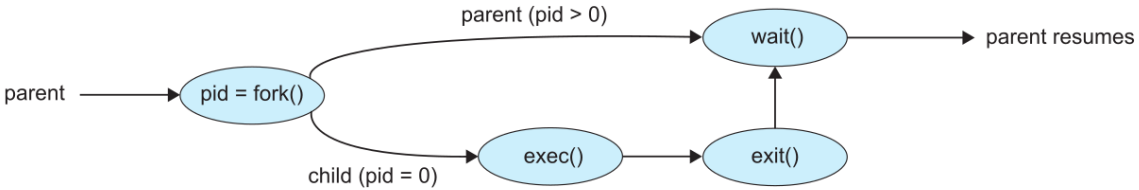
Creación de procesos

```
#include <unistd.h>   //fork()
#include <sys/wait.h> //wait()
#include <iostream>
using namespace std;

int main() {
    pid_t pid = fork();                    // fork a child process.

    if (pid < 0) {
        cout << "Fork Failed" << endl;
        return 1;
    } else if (pid == 0) {                 // child process because return value zero.
        cout << "Child process..." << endl;
        execlp("/bin/ls", "ls", NULL);
    } else {                               // parent process because return value non-zero (positive).
        wait(NULL);
        cout << "Parent process..." << endl;
    }
    return 0;
}
```

Creación de procesos



Terminación de procesos

- Proceso termina cuando ejecuta su última instrucción y pide al SO que lo elimine mediante la llamada a sistema **exit()**.
- Proceso hijo puede devolver su estado al Padre mediante **wait(&status)**.
- SO libera los recursos asignados al proceso.
- Proceso Padre puede terminar la ejecución de uno de sus hijos:
  - Hijo excede uso de recursos asignados.
  - La tarea asignada al Hijo ya no es necesaria.
  - Padre abandona el sistema, el SO no permite que procesos Hijos continúen si su padre ya ha terminado.

UNIVERSIDAD DEL BÍO-BÍO

Comunicación Interprocesos

- Procesos que se ejecutan concurrentemente pueden ser:
  - **Independientes**: no comparte datos con otros procesos en el sistema.
  - **Cooperativos**: puede afectar o verse afectado por los demás procesos. (Cualquier proceso que comparte datos con otros es Cooperativo).
- Hay varias razones para permitir cooperación entre procesos:
  - Compartir información. Varios usuarios pueden estar interesados en la misma información (archivo compartido).
  - Acelerar los cálculos. Dividir tareas en subtareas. Ejecución en paralelo (varias CPUs).
  - Modularidad. Construcción de un sistema modular, dividiendo las funciones del sistema en diferentes procesos.

UNIVERSIDAD DEL BÍO-BÍO

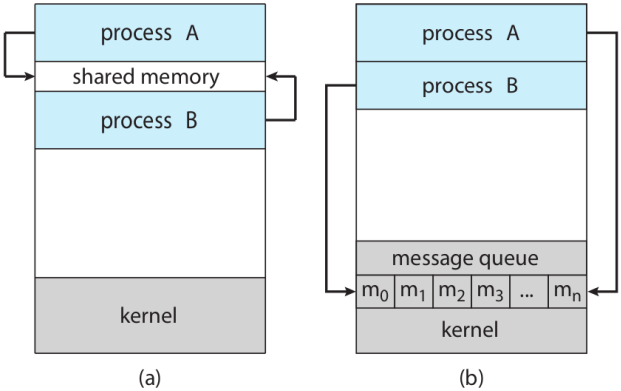
Comunicación Interprocesos

- La cooperación entre procesos necesita de mecanismos de **Comunicación Interprocesos** (IPC, interprocess communication) que permitan intercambiar datos entre ellos.
- Existen dos modelos fundamentales:
  - **Memoria Compartida**.
    - Establece región de la memoria para que sea compartida por los procesos cooperativos.
    - Intercambian información leyendo/escribiendo datos en tal zona.
    - Permite velocidad máxima (velocidades de la memoria)
  - **Paso de Mensajes**:
    - Comunicación tiene lugar mediante intercambio de mensajes entre procesos cooperativos.
    - Más complejo de implementar.
    - Más lenta ya que usa llamadas al sistema, por lo que requiere intervención del kernel (consume más tiempo).

UNIVERSIDAD DEL BÍO-BÍO

Comunicación Interprocesos

- (a) Memoria Compartida (b) Paso de Mensajes.



Memoria Compartida - APIs

- IEEE POSIX Threads (PThreads)[1]:
  - Standard UNIX threading API (disponible en Windows).
  - Más de 60 funciones: pthread_create, pthread_join, pthread_exit, etc.
  - Relativamente de bajo nivel.
  - Manejo explícito de las hebras (creación, eliminación de hebras, sincronización).

[1] POSIX Threads Programming: `https://computing.llnl.gov/tutorials/pthreads/`

Memoria Compartida - APIs

- OpenMP:
  - Extensión de lenguaje basado en (directivas de compilación, biblioteca de funciones.)
  - Disponible para C/C++ y Fortran.
  - Énfasis en high-performance computing (HPC).
  - Alto nivel de abstracción.
  - OpenMP: `https://www.openmp.org/`

```
#pragma omp parallel for
for( i = 0; i < n; i++ )
  z[i] = a * x[i] + y[i]
```

UNIVERSIDAD DEL BÍO-BÍO

Paso de Mensajes - API

- Múltiples procesos ejecutándose en diferentes nodos (pueden ser locales también).
- Cada proceso con su espacio de direcciones de memoria privado.
- Acceso a datos (remotos) de otros procesos via envío/recepción de mensajes.
- MPI (Message Passing Interface) es el estandar *de-facto*.
- OpenMPI: `https://www.open-mpi.org/`

```
if (process_id == SENDER)
  send_to(RECEIVER, data);

if (process_id == RECEIVER)
  recv_from(SENDER, data);
```

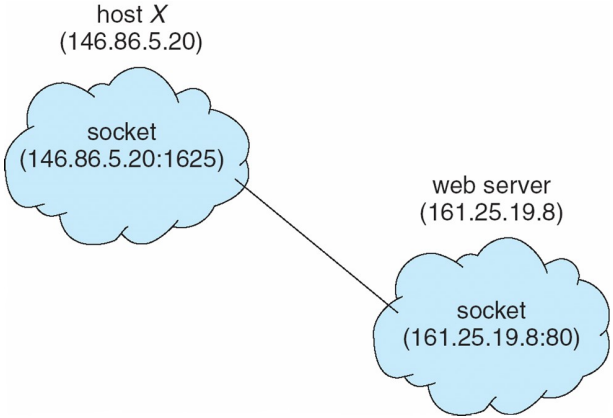
UNIVERSIDAD DEL BÍO-BÍO

Sockets

- Hasta el momento hemos descrito como pueden comunicarse los procesos utilizando las técnicas de memoria compartida y paso de mensaje.
- Estás técnicas también pueden emplearse en los sistemas C-S para establecer comunicaciones.
- **Sockets**:
  - Par de procesos emplean una pareja de sockets para comunicarse a través de una red de datos.
  - Se identifican mediante dirección IP concatenada con un número de puerto.
  - Servidores implementan servicios que **escuchan** en puertos conocidos (80→HTTP, 443→HTTPs, 22→SSH, 21→FTP, etc).
  - Cliente X **146.86.5.20:1625** tiene una conexión establecida con el Servidor WEB **161.25.19.8:80**

UNIVERSIDAD DEL BÍO-BÍO

Sockets


host X
(146.86.5.20)
socket
(146.86.5.20:1625)
web server
(161.25.19.8)
socket
(161.25.19.8:80)