CC4302
Sistemas Operativos
Profesor: Luis Mateu

Unidad 2: administración de procesos

- Principio de virtualización
- Procesos livianos vs. procesos pesados
- Preemptiveness
- Scheduling de procesos
- Interrupciones y timer
- Estados de un proceso
- Descriptor de proceso
- Ráfagas de CPU

Unidad 2: Administración de procesos

El núcleo del sistema operativo

- Brinda los servicios básicos para que los procesos se puedan ejecutar, como por ejemplo las llamadas a sistema (*open, read, fork, pipe,* etc.)
- Siempre residente en memoria
- Se ejecuta en modo privilegiado: tiene acceso a toda la máquina, sin restricciones
- Asigna de manera eficiente los recursos de hardware a los procesos

Un proceso

- Es un procesador virtual en donde se ejecuta una aplicación (*Chrome, Gimp, LibreOffice*) o herramienta del sistema operativo (*ls, cp, zip*)
- Se ejecuta en un modo protegido: no tiene acceso directo a la máquina, pero está protegido de las acciones de otros procesos

Principio de virtualización

- Para asignar eficientemente un recurso limitado, el sistema operativo lo transforma en múltiples recursos virtuales idénticos, otorgables a diferentes usuarios
- Ejemplo 1: el hardware ofrece solo *p* cores, entonces el sistema permite crear múltiples cores virtuales denominados *threads* (o procesos livianos) que comparten la misma memoria
- Ejemplo 2: el hardware ofrece una sola memoria, entonces el sistema permite crear múltiples procesadores virtuales con su propia memoria, denominados procesos Unix o procesos pesados
- Ejemplo 3: el hardware ofrece una sola pantalla, entonces el sistema gráfico permite crear múltiples pantallas virtuales denominadas ventanas
- Ejemplo 4: el hardware ofrece unos pocos discos de tamaño fijo, entonces el sistema permite crear múltiples discos virtuales de tamaño variable denominados archivos
- Otros ejemplos: la red, la impresora, el cronómetro

Clasificación de procesos: livianos vs. pesados

Según si comparten o no la memoria: ***threads*** sí la comparten, ***procesos Unix*** si no la comparten

- Los primeros procesadores no tenían la *Memory Management Unit* necesaria para implementar procesos pesados y por lo tanto ofrecían solo threads, por ejemplo el commodore Amiga 1000 (1985)

- Si las aplicaciones corren en threads, pueden interferir entre ellas y por lo tanto no hay protección
- Si una aplicación se caía en el Amiga había que dar reset

- Los procesos pesados otorgan protección pero son más costosos en cpu y uso de memoria

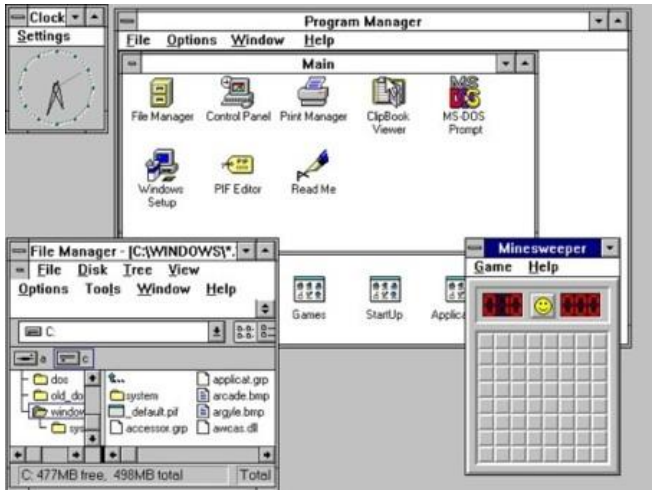
Clasificación de procesos: preemptiveness

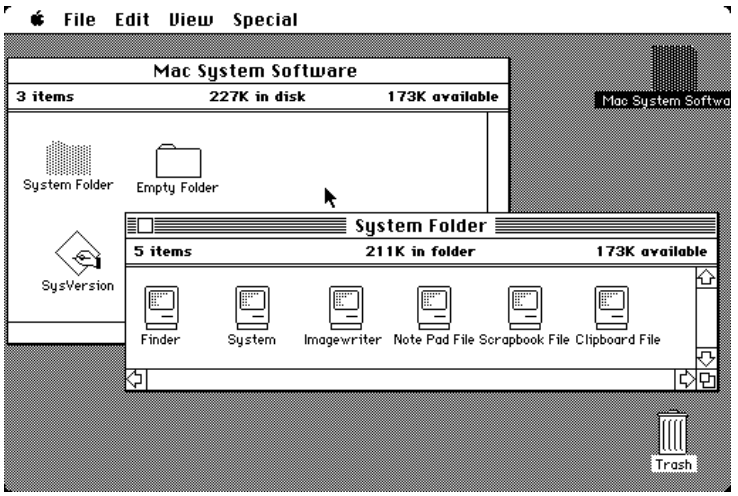
Según si el núcleo puede quitarles el core que ocupan o no: a un ***proceso preemptive*** sí se le puede quitar, no a un proceso ***non preemptive*** (sin adelantamiento)

- Una vez que el núcleo le cede un core a un proceso non preemptive, solo el mismo proceso se lo puede devolver al núcleo
- Los primeros sistemas que fueron capaces de cargar simultánemante múltiples aplicaciones solo permitían procesos non preemptive porque si se les quitaba el core dejaban de funcionar establemente
- La aplicación debía devolver rápidamente el core para tener un buen *tiempo de respuesta*
- Si una aplicación se quedaba en un ciclo infinito, había que dar reset

Ejemplos de sistemas preemptive y non preemptive

- Windows 3.11 y los primeros Mac Os solo ofrecían procesos livianos non preemptive





- En Unix/Linux/Android/Windows/OS X todos los procesos son preemptive y pueden ser pesados o livianos

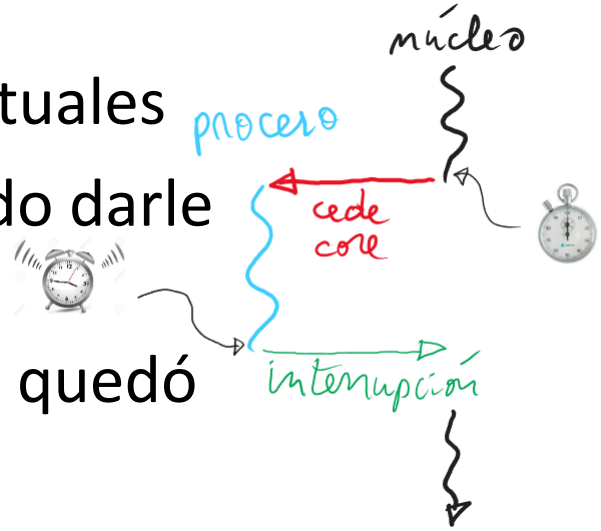
Scheduling de procesos

- En este capítulo vamos a estudiar la virtualización de los cores, es decir la implementación de los threads
- La atribución estratégica de los cores a los distintos threads en ejecución se denomina ***scheduling de procesos***
- De ello se encarga el scheduler de procesos en el núcleo
- Se busca maximizar alguna propiedad particular como por ejemplo el tiempo de despacho o el tiempo de respuesta

Interrupciones y timer

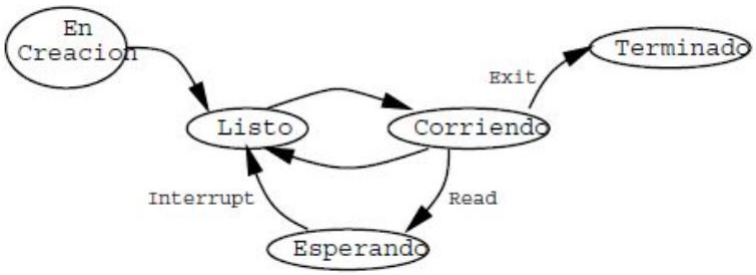
- Para poder implementar procesos preemptive, el procesador debe admitir *interrupciones* e incluir un *cronómetro regresivo* (*timer*) por cada core

- Antes de ceder un core a un proceso, el scheduler configura el timer para gatillar una interrupción después de unas pocas centésimas de segundo
- La interrupción le devuelve el core al núcleo
- Las interrupciones son como las señales de PSS, pero se envían a los drivers en el núcleo, no a un proceso
- Las señales son interrupciones virtuales
- Más tarde, el núcleo decide cuando darle nuevamente un core al proceso
- El proceso no se da cuenta que se quedó sin core por un momento



Estados de un proceso

- Mientras se ejecutan los procesos, transitan por diversos estados



- En creación: núcleo trabajando para obtener los recursos del proceso
- RUN: el proceso se ejecuta en algún core
- WAIT: el proceso espera algún evento, como la lectura de un sector del disco
- READY: no se le ha asignado ningún core al proceso pero es elegible para uno
- ZOMBIE: el proceso terminó pero espera ser enterrado

El descriptor de proceso

- Es una estructura de datos en donde el núcleo mantiene información asociada a un proceso:
  - Su estado
  - Registros del procesador cuando no tiene un core asignado
  - Información de scheduling: prioridad, siguiente proceso en la cola de espera, etc.
  - Asignación de recursos: memoria asignada, archivos abiertos, espacio de swapping, etc.
  - Contabilización de uso de recursos: tiempo de CPU
- Colas de scheduling: en donde aguardan los descriptores de proceso a la espera de recursos
- Identificador de proceso (pid): identificador público de un proceso, típicamente un entero entre 0 y 99999
- Cambio de contexto: cuando el scheduler traspasa un core de un proceso a otro
  - Es costoso, especialemente entre procesos pesados
  - Implícitos: el proceso no cedió voluntariamente el core, el núcleo se lo quitó, el scheduler los puede evitar
- No confundir con interrupciones

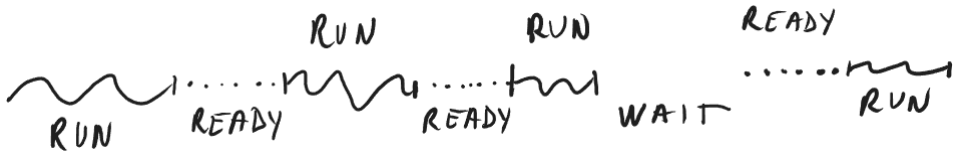
Ejemplo: nThreads

- Es un sistema operativo con fines pedagógicos
- Se ejecuta en un proceso de Linux con un número fijo de pthreads
- Los pthreads se virtualizan en un sin número de nano threads
- Instructivo agregar nuevos schedulers o nuevas herramientas de sincronización
- La misma API de los pthreads
- El descriptor de un nano thread:

```
typedef struct nthread nThread;

struct nthread {
   State status;            // RUN, READY, ZOMBIE, etc.
   char *thread_name;       // Useful for debugging purposes

   void *queue;             // The queue where this thread is waiting
   nThread nextTh;          // Next node in a linked list of threads
   nThread nextTimeTh;      // Next node in a time ordered linked
   void **sp;               // Thread stack pointer when suspended
   void **stack;            // Pointer to the whole stack area

   // For nThreadExit and nThreadJoin
   void *retPtr;
   nThread joinTh;

   int wakeTime;            // For time queues
};
```

Ráfagas de CPU

- Secuencia de instrucciones de un proceso sin pasar por estados de espera
- Un proceso se ejecuta en muchas ráfagas
- Proceso intensivo en CPU: sus ráfagas duran mucho tiempo
- Proceso intensivo en E/S: sus ráfagas son cortas
- El scheduler puede ejecutar las ráfagas largas en varias tajadas de tiempo para darle oportunidad a otros procesos de ejecutarse



- Histograma de duración:
- Muchas ráfagas cortas
- Muy pocas de larga duración