Tema

5.1

Sistema Operativos (SSOO)

Gestión de memoria

Universidad Francisco de Vitoria
UFV Madrid

Grado en Ingeniería Informática
Escuela Politécnica Superior

Índice

- Memoria
- Programas de usuario
- Gestión de memoria

Memoria

- Una gestión efectiva de la memoria es vital
- El objetivo es repartir eficientemente la memoria para cargar tantos procesos como sea posible
- De esta forma se conseguirá maximizar el uso del procesador.

Tipos de memoria:

- **Memoria principal**, o RAM – volátil pero de acceso rápido.
  - No proporciona un almacenamiento permanente
- **Memoria secundaria** – memoria de disco duro.
  - Es más lenta y no volátil, así que ofrece almacenamiento a largo plazo.

Memoria

- Existe un trasvase entre ambos tipos de memoria, tanto de datos como de código.
- Gestionar este intercambio es tarea del sistema operativo.
- Un programa debe cargarse en memoria desde disco y colocarse en el procesador (**como proceso**) para que se ejecute.
- La memoria principal (RAM) y los registros del procesador son los únicos dispositivos de almacenamiento a los que puede acceder la **CP**U directamente.

Memoria

- Existe un trasvase entre ambos tipos de memoria, tanto de datos como de código.
- Gestionar este intercambio es tarea del sistema operativo.
- Un programa debe cargarse en memoria desde disco y colocarse en el procesador (**como proceso**) para que se ejecute.
- La memoria principal (RAM) y los registros del procesador son los únicos dispositivos de almacenamiento a los que puede acceder la **CP**U directamente.
  - Un registro es una memoria de alta velocidad y poca capacidad, integrada en el procesador, que permite guardar transitoriamente y acceder a valores muy usados.
  - Los registros del procesador se usan para acceder a instrucciones de procesos en ejecución

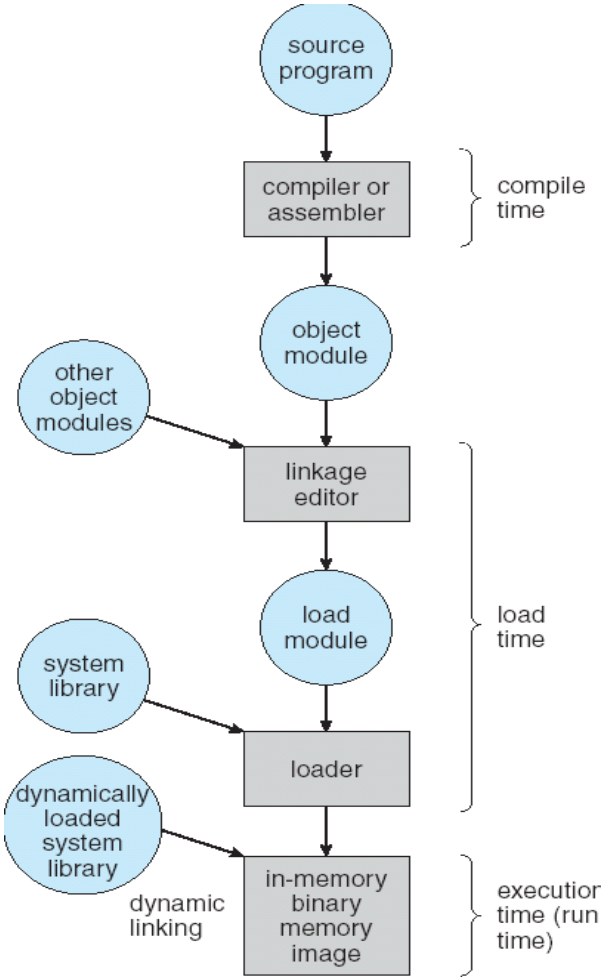
Memoria

- El acceso a registro es muy rápido; supone un ciclo de CPU (o menos) →(Hz)
- El acceso a memoria principal puede durar varios ciclos.
- Las memorias caché se colocan entre la memoria principal y la CPU para acelerar el acceso a la información.
- Esta parte más de hardware se explica también en arquitectura de ordenadores.

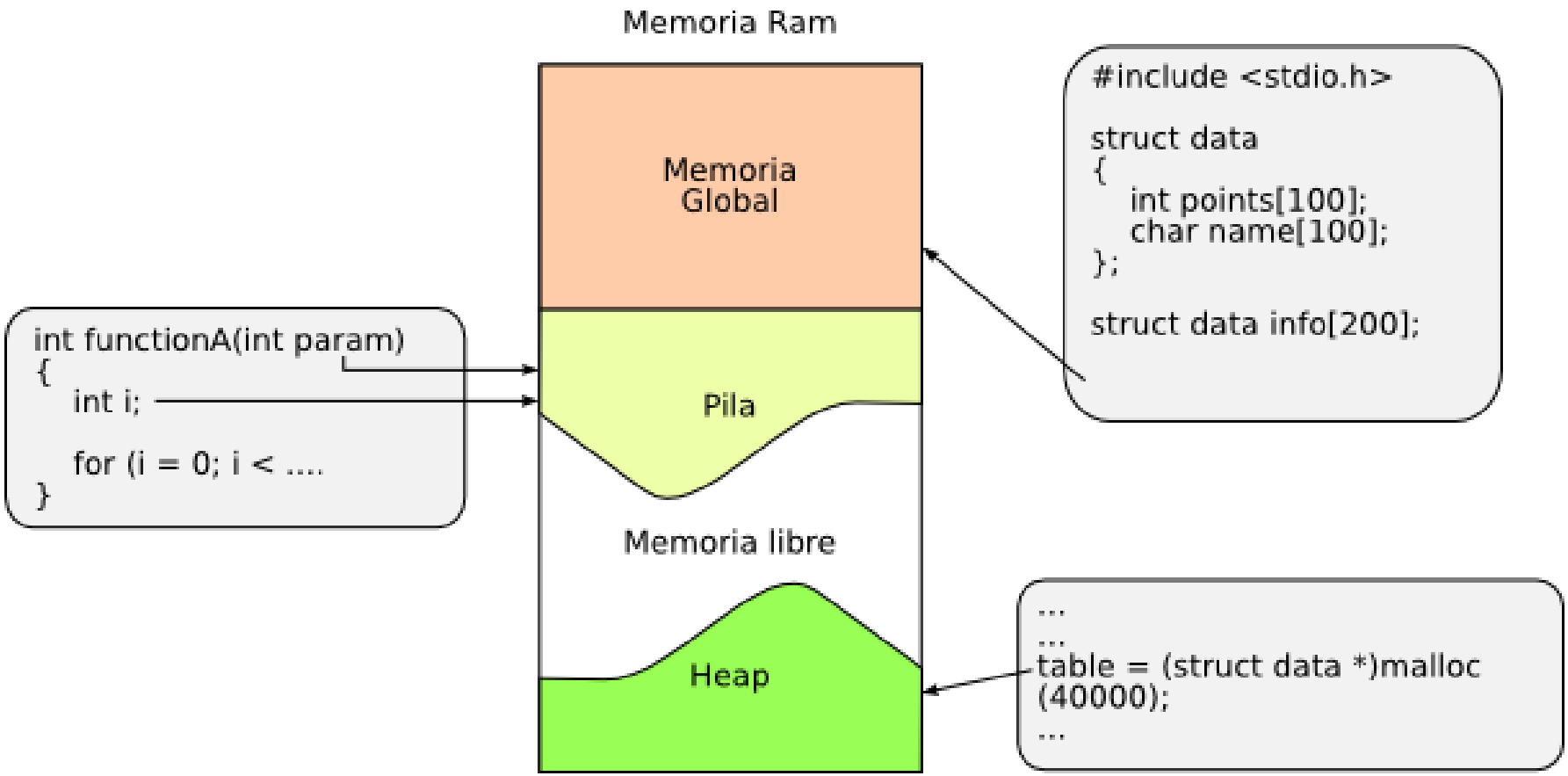
Programas de usuario

Para que un ordenador pueda ejecutar la interpretación de un programa escrito en un lenguaje de alto nivel se siguen cuatro etapas diferentes:

- Compilación
- Enlace o linkado
- Carga (load)
- Ejecución



Programas de usuario



Uso de la memoria: en c

Un programa en C almacena sus datos en memoria en tres áreas diferentes:

- Memoria global. Es el área en la que están almacenadas las variables que se declaran globales o estáticas y las constantes de tipo cadena de caracteres (por ejemplo "Mi string"). Es decir, en esta zona de memoria se almacenan todos aquellos datos que están presentes desde el comienzo del programa hasta que termina.
- La pila. Es un área en la que las variables aparecen y desaparecen en un momento puntual de la ejecución de un programa. Se utiliza principalmente para almacenar variables locales a las funciones. Estas variables tienen un ámbito reducido, sólo están disponibles mientras se está ejecutando la función en la que han sido definidas. En la pila se encuentran todas estas variables, y por tanto, en esa zona se está continuamente insertando y borrando variables nuevas.
- El heap. Esta zona (traducida en algunos casos como “el montón”) contiene memoria disponible para que se reserve y libere en cualquier momento durante la ejecución de un programa. No está dedicada a variables locales de las funciones como la pila, sino que es memoria denominada “dinámica” para estructuras de datos que no se saben si se necesitan, e incluso tampoco se sabe su tamaño hasta que el programa está ejecutando.

Uso de la memoria: en c

Las cuatro operaciones principales para gestionar memoria en C son:

- void *malloc(size_t size). Es la función para reservar tantos bytes consecutivos de memoria como indica su único parámetro. Devuelve la dirección de memoria de la porción reservada. La memoria no se inicializa a ningún valor.
- void *calloc(size_t nmemb, size_t size). Reserva espacio para tantos elementos como indica su primer parámetro nmemb, y cada uno de ellos con un tamaño en bytes como indica el segundo. En otras palabras, reserva nmemb * size bytes consecutivos en memoria. Al igual que la función anterior devuelve la dirección de memoria al comienzo del bloque reservado. Esta función inicializa todos los bytes de la zona reservada al valor cero.
- void free(void *ptr). Función que dado un puntero, libera el espacio previamente reservado. El puntero que recibe como parámetro esta función tiene que ser el que se ha obtenido con una llamada de reserva de memoria. No es necesario incluir el tamaño. Una vez que se ejecuta esta llamada, los datos en esa porción de memoria se consideran basura, y por tanto pueden ser reutilizados por el sistema.
- void *realloc(void *ptr, size_t size). Función para redimensionar una porción de memoria previamente reservada a la que apunta el primer parámetro al tamaño dado como segundo parámetro. La función devuelve la dirección de memoria de esta nueva porción redimensionada, que no tiene por qué ser necesariamente igual al que se ha pasado como parámetro. Los datos se conservan intactos en tantos bytes como el mínimo entre el tamaño antiguo y el nuevo..

Programas de usuario: compilación

- Por regla general, el compilador no produce directamente un fichero ejecutable, sino que el código generado se estructura en módulos que se almacenan en un fichero objeto.
- Los ficheros objeto poseen información relativa tanto al código máquina como a una tabla que almacena la estructura de las variables y tipos utilizados por el programa fuente.

Programas de usuario: enlazador

- Un enlazador es un programa que toma los objetos generados en los primeros pasos del proceso de compilación, la información de todos los recursos necesarios (librerías), quita aquellos recursos que no necesita, y enlaza el código objeto con su(s) librería(s) con lo que finalmente produce un fichero ejecutable (o una librería).

Programas de usuario: carga

- Cuando el enlazador (linker) construye el fichero ejecutable, asume que cada segmento va a ser colocado en la dirección cero de la memoria.
- Es una dirección lógica.
- Como el programa va a estar dividido en segmentos (ya lo veremos), las direcciones a que hacen referencia las instrucciones dentro de cada segmento (instrucciones de cambio de control de flujo, de acceso a datos, etc.), no se tratan como absolutas.
- Lo que se hace es tratarlas como direcciones relativas a partir de la dirección base en que sea colocado cada segmento en el momento de la ejecución.

Programas de usuario: carga

- El cargador carga el fichero .exe (o binario), coloca sus diferentes segmentos en memoria (donde el sistema operativo le diga que hay memoria libre para ello)
- Y asigna después los registros base (direcciones primeras) a sus posiciones correctas (direcciones físicas), de manera que las direcciones relativas funcionen correctamente.
- El fichero ejecutable tiene una cabecera y unas secciones que permiten construir los segmentos de la imagen de memoria del proceso.

Programas de usuario: mapa de memoria

- El sistema operativo tiene una visión del proceso consistente en un conjunto de regiones. En sistemas con memoria virtual, las diferentes regiones que integran el proceso suelen estar separadas y tienen un tamaño de un número entero de páginas.
- Las regiones más relevantes de la imagen de memoria de un proceso son:
  - Código (texto).- que contiene el código máquina del programa.
  - Datos:
    - Con valor inicial
    - Sin valor inicial
    - Datos creados dinámicamente o Heap
  - Pila.

Programas de usuario: mapa de memoria

- El modo en el que se estructuran las regiones depende del diseño del sistema operativo.
- Vamos a comenzar el análisis de las regiones de memoria de un proceso a través de la información proporcionada por el directorio /proc.
- Para ello vamos a utilizar un programa escrito en C y además vamos a compilarlo estáticamente, es decir, sin bibliotecas dinámicas. Para ello utilizaremos la opción **-static** del mandato gcc.
- A continuación, podemos obtener el mapa de memoria de memoria del proceso, accediendo al fichero maps del directorio /proc/<pid>:

```
ll]
mcardenas@mcardenas:~$ clear
mcardenas@mcardenas:~$ cat /proc/2013/maps
55c4a26b0000-55c4a26b1000 r-xp 00000000 08:02 402223                     /home/mcardenas/hola
55c4a28b0000-55c4a28b1000 r--p 00000000 08:02 402223                     /home/mcardenas/hola
55c4a28b1000-55c4a28b2000 rw-p 00001000 08:02 402223                     /home/mcardenas/hola
55c4a397d000-55c4a399e000 rw-p 00000000 00:00 0                          [heap]
7f190b6a5000-7f190b88c000 r-xp 00000000 08:02 525076                     /lib/x86_64-linux-gnu/libc-2.27.so
7f190b88c000-7f190ba8c000 ---p 001e7000 08:02 525076                     /lib/x86_64-linux-gnu/libc-2.27.so
7f190ba8c000-7f190ba90000 r--p 001e7000 08:02 525076                     /lib/x86_64-linux-gnu/libc-2.27.so
7f190ba90000-7f190ba92000 rw-p 001eb000 08:02 525076                     /lib/x86_64-linux-gnu/libc-2.27.so
7f190ba92000-7f190ba96000 rw-p 00000000 00:00 0
7f190ba96000-7f190babd000 r-xp 00000000 08:02 525064                     /lib/x86_64-linux-gnu/ld-2.27.so
```

Programas de usuario: mapa de memoria

El proceso dispone de 6 regiones de memoria, todas ellas privadas, puesto que tienen activo el bit p en los permisos.

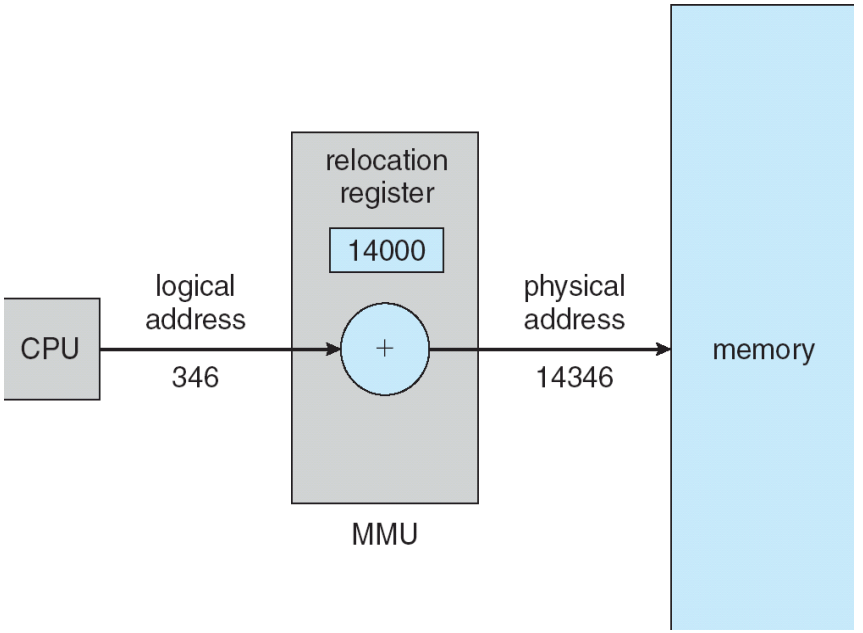
- Segmento 1.- Esta región tiene activos los permisos de lectura y ejecución y puesto que su contenido se obtiene del propio fichero ejecutable, podemos deducir que esta región contiene el código (texto) del proceso. Su tamaño es de ¿? = ¿? (¿? bytes)
- Segmento 2.- Esta región tiene activos los permisos de lectura y escritura y puesto que su contenido se obtiene del propio fichero ejecutable, podemos deducir que esta región contiene los datos del programa, más concretamente, los datos con valor inicial. Su tamaño es de: ¿? = ¿? B.
- Segmento 3.- Este tercer segmento, es propio y dependiente del sistema operativo en el que se ejecuta el proceso en cuestión. En otro sistema operativo no tiene porqué existir. En este caso se trata de una zona de 12 Kb que es residente en memoria RAM.

Programas de usuario: mapa de memoria

- Segmento 4.- Como su propio nombre indica, esta región corresponde al heap del proceso. Su tamaño es de ¿? B.
- Segmento 5.- Como su propio nombre indica, se trata de la pila del proceso. Su tamaño es de ¿? B.
- Segmento 6.- El sexto segmento corresponde a una región que se estudiará en un curso más avanzado, por lo que no se incide ahora en ella. Simplemente indicar que se trata de una región de memoria común a todos los procesos de sistema.
- El directorio /proc/<pid>/ contiene un fichero en el que se muestra de manera más detallada la información sobre los segmentos de memoria de un proceso, en concreto, podríamos conocer cuanta información del segmento tienen soporte en memoria RAM.

Gestión de memoria

- La unidad de gestión de memoria o MMU (del inglés Memory Management Unit) es un dispositivo de hardware responsable de la gestión de los accesos a la memoria RAM por parte de la Unidad de Procesamiento Central (CPU) o procesador.
- Al final traduce direcciones de memoria lógicas a físicas.



Una buena gestión de memoria tiene que asegurar que se cumplan dos requisitos imprescindibles: Reubicación y Protección

Gestión de memoria: reubicación

- El objetivo es poder cargar y descargar los procesos activos en la memoria principal (RAM) para maximizar el uso del procesador.
- ¿Os acordáis de la planificación? Pues aquí van los procesos que están en espera o listos
- De esta forma se mantiene una gran reserva de procesos listos para ejecutarse a los que podremos acceder rápidamente para volver a cargarlos en el procesador.
- Una vez que se ha descargado el proceso en memoria, cuando vuelve a ser cargado no se vuelve a cargar en la misma región de memoria
- Lo que se hace es reubicarlo en un área distinta de memoria.
- Acordaos también que cuando se expulsó a un proceso del procesador, se realizó un cambio de contexto por el que se apuntó en el BCP el estado del proceso y los datos y la instrucción en la que se quedó cuando fue expulsado.

Gestión de memoria: protección

- Cada proceso debe protegerse contra interferencias no deseadas de otros procesos, tanto accidentales como intencionadas.
- Así pues, el código de un proceso no puede hacer referencia a posiciones de memoria de otros procesos, con fines de lectura y escritura, sin permiso
- Con lo cual al final está muy relacionado con la reubicación
- Esto al final es tarea del hardware (procesador).

Bibliografía

- **CARRETERO**, Jesús, GARCÍA, Félix, DE MIGUEL, Pedro, PÉREZ, Fernando. Sistemas Operativos: una visión aplicada. McGraw-Hill, 2001.
- **STALLINGS, William. Sistemas operativos: aspectos internos y principios de diseño. 5ª Edición. Editorial Pearson Educación. 2005. ISBN: 978-84-205-4462-5.**
- **TANENBAUM**, Andrew S. Sistemas operativos modernos. 3ª Edición. Editorial Prentice Hall. 2009. ISBN: 978-607- 442-046-3.

marlon.cardenas@ufv.es

Universidad Francisco de Vitoria
UFV Madrid

Grado en Ingeniería Informática
Escuela Politécnica Superior