

3.1

a) Para establecer la tabla de páginas del proceso de usuario, primero debemos analizar los segmentos del proceso y calcular cuantas páginas se necesitan por cada uno:

- Segmento de texto: $\frac{3,4KB}{4KB} \approx 1$ página

- Segmento de pila: $\frac{200B}{4KB} \approx 1$ página

- Segmento de datos de solo lectura: $\frac{14KB}{4KB} = 3's \approx 4$ páginas

- Segmento de datos de escritura y lectura: $\frac{6KB}{4KB} = 1'5 \approx 2$ páginas

Nº página	Segmento	Dirección Virtual	Dirección Física	En Memoria Principal	En Swap
0	Texto	0x0000	0x1000	Sí	NC
1	pila	0x1000	0x2000	Sí	No
2	Datos RO	0x2000	0x3000	Sí	NC
3	Datos RO	0x3000	0x4000	Sí	No
4	Datos RC	0x4000	0x5000	Sí	NC
5	Datos RC	0x5000	-	No	Sí
6	Datos RW	0x6000	0x6000	Sí	No
7	Datos RW	0x7000	-	No	Sí

5) • Memoria Principal:

- SO(4KB) ocupando las primeras 4KB
- 1 página del segmento de texto
- 1 página del segmento de pila
- 3 páginas del segmento de datos de solo lectura
- 1 página del segmento de datos de escritura y lectura.

• Zona de swap:

- 1 página del segmento de datos de solo escritura
- 1 página del segmento de datos de escritura y lectura.

c) Se producirá un desbordamiento de pila (stack overflow). El sistema operativo deberá asignar una nueva página de memoria para ampliar la pila. Dado que la memoria principal ya está llena, el SO tendrá que liberar una página de algún segmento en la memoria principal y moverla a la zona de swap.
En este caso, el sistema operativo podría seleccionar una de las páginas menos utilizadas y moverla a la zona de swap, liberando espacio en la memoria principal. Luego, asigne una nueva página.

3.2

a)

Cabecera (1KB)	Código (5KB)	Datos con valor inicial (d.v.i) (6KB)	Datos sin valor inicial (d.s.v.i) (2KB)
-------------------	-----------------	---	---

Al ejecutar el fichero se tendrá un tamaño de 14KB.

b)

Código (5KB)	d.v.i (6KB)	d.s.v.i (2KB)	lib.so
X [C24] (1KB)	Y [I024] (1KB)	Pila	Variables locales

c) Si el sistema de gestión de memoria inicial se genera fragmentación interna, que ocurre cuando el espacio de memoria se divide en bloques fijos (en este caso, páginas) y el tamaño de los bloques no se ajusta exactamente al tamaño de los segmentos que se almacenan en ellos.

La fragmentación se puede observar en el catálogo inicial del disco, donde se muestra que los bloques que contienen los segmentos de lib.so y los segmentos de texto y datos no están completamente llenos.

3.3

a) Tabla de páginas de primer nivel:

Índice	Entrada de Tabla de Páginas (ETP)
0	Dirección de la tabla de 2º nivel

Tabla de páginas de segundo nivel:

Índice	Valido	Presente	Modificado	Referenciado	COW	RC	RF	Protección	Marc/Bloque
0	1	1	0	1	0	0	0	R-X	0xccc
1	1	1	0	1	0	0	0	RW-	0x001
2	1	1	0	1	0	0	0	RW-	0xcc2
3	1	1	0	1	0	0	0	RW-	0xcc3
4	1	1	0	1	0	0	0	RW-	0xcc4
5	1	1	0	1	0	0	0	RW-	0xcc5

b)

- Si una misma página es expulsada varias veces al área swap, esto indica que la página está siendo frecuentemente reemplazada y vuelve a cargar en la memoria principal, lo que prueba que las páginas son expulsadas y cargadas repetidamente. Esto puede causar una degeneración en el rendimiento del sistema debido a la alta tasa de intercambio de páginas.

2. El bit modificado se escribe por el hardware (MMU) cuando se realiza una escritura en la página y es inicializado por el SO cuando la página se carga en la memoria principal o se recupera del disco. Este bit es crucial para asegurar una gestión eficiente de la memoria y reducir las escrituras innecesarias en el disco.

3.4

a) Al llegar el proceso al instante B, se han ejecutado el for y la función func. Esta función necesita solamente la primera página de la región de texto y la primera página de la región de datos de lib.so. Por lo que estas páginas de la biblioteca deben ser mapeadas. Así el mapa de memoria sería:

Código (5KB)	d.v.i (6KB)	d.s.v.i (2KB)	lib.so Texto (4KB)
x [1024] (1KB)	y [1024] (1KB)	Pila	Variables Locales
lib.so			
Datos (4KB)			

Suponiendo que las páginas de la biblioteca dinámica se cargan en los marcos de página 0x006 y 0x007, la tabla de páginas de segundo nivel sería:

Índice	Valido	Presente	Modificado	Referenciado	COW	RC	RF	Protección	Mosca/Bloque
0	1	1	0	1	0	0	0	R-X	0x0000
1	1	1	0	1	0	0	0	RW-	0x001
2	1	1	0	1	0	0	0	RW-	0x002
3	1	1	0	1	0	0	0	RW-	0x003
4	1	1	0	1	0	0	0	RW-	0x004
5	1	1	0	1	0	0	0	RW-	0x005
6	1	1	0	1	0	0	0	R-X	0x006
7	1	1	0						

b) Cuando un proceso crea un proceso hijo, generalmente se utiliza la técnica de COW para compartir las páginas de memoria entre el proceso padre y el hijo, y se marca el bit COW en los entradas de la tabla de páginas correspondientes.

Cuando el proceso hijo intenta escribir en la zona de datos de la biblioteca dinámica, el hardware detectará que el bit COW está activado para esa página de memoria. En respuesta, el sistema operativo creará una copia de la página de memoria y asignará la nueva copia al proceso hijo, actualizando la entrada correspondiente en la tabla de páginas del proceso hijo. El proceso hijo escribirá en su propia copia de la página de memoria, mientras que el proceso padre seguirá utilizando la versión original de la página de memoria.

3.5

a)

Código	Datos Inicial	Datos No Inicial	BSS
(3.573 bytes)	(4 bytes) b	(4 bytes) a	(4 bytes) e

- Código: Contiene el código ejecutable del programa, incluidos el main y función.
- Datos Inicializados: contiene las variables globales inicializadas (b)
- Datos No Inicializados: contiene variables no inicializadas (a).
- BSS (Bloque de Inicio de Almacenamiento) : contiene variables no inicializadas y variables estáticas no inicializadas , (e)

b)

Código	Datos Inicial	Datos No Inicial	BSS	Pila
(3.573 bytes)	(4 bytes) b	(4 bytes) a	(4 bytes) e	
Datos Libreria libc (simbólico)				

- Código: desde la dirección de inicio hasta la dirección de inicio + 3573 bytes . Contiene el código ejecutable del programa.

- Datos Inicializados: desde la dirección de inicio del código + 3.573 B hasta la dirección de inicio del código + 3.573 bytes + 4 bytes. Contiene la variable global inicializada b con el valor 50.
- Datos No Inicializados: desde la dirección de inicio de los datos inicializados + 4 bytes hasta la dirección de inicio de los datos inicializados + 4 bytes + 4 bytes. Contiene la variable global no inicializada a.
- BSS: desde la dirección de inicio de los datos no inicializados + 4 bytes hasta la dirección de inicio de los datos no inicializados + 4 bytes + 4 bytes. Contiene la variable estática no inicializada e con el valor 0.
- Pila: la región pila está vacía en este punto, ya que aún no se han llamado funciones ni se han creado variables locales.

Además, hay una región para los datos de libC, pero como no se especifican sus necesidades de memoria, se representan mediante un nombre simbólico.

3.6

Cuando el programa ejecuta hasta las líneas 12 y 21, se han llamado a las funciones `function` y `malloc`, lo que afecta el estado de la pila y el heap. Hugo el gráfico sería el siguiente:

Código	Datos Inicial	Datos No Inicial	BSS	Heap	Pila
(3.573 bytes)	(4 bytes) $b = 50$	(4 bytes) a	(4 bytes) $e = 2$	(5.120 bytes) f	
Data librería libc (smbCilic)					Variables locales c, d

- Código: desde la dirección de inicio hasta la dirección de inicio + 3.573 bytes. Contiene el código ejecutable del programa.
- Datos Inicializados: desde la dirección de inicio del código + 3.573 B hasta la dirección de inicio del código + 3.573 bytes + 4 bytes. Contiene la variable global inicializada b con el valor 50.
- Datos No Inicializados: desde la dirección de inicio de los datos inicializados + 4 bytes hasta la dirección de inicio de los datos inicializados + 4 bytes + 4 bytes. Contiene la variable global no inicializada a .

- BSS: desde la dirección de inicio de los datos no inicializados + 4 bytes hasta la dirección de inicio de los datos no inicializados + 4 bytes + 4 bytes. Contiene la variable entérica no inicializada e con el valor 0.
- Heap: desde la dirección de inicio del heap hasta la dirección de inicio del heap + 5.120 bytes. Contiene la memoria asignada dinámicamente para g mediante malloc en la línea 19.
- Pila: contiene las variables locales c y d de función g() están siendo ejecutado.

Además, hay una región para los datos de libc, pero como no se especifican sus necesidades de memoria, se representan mediante un nombre simbólico.

3.13

- a) 1. Leer el archivo ejecutable xxxx del sistema de archivos y analizar su estructura, como las secciones de texto, datos, etc.
2. Crear una nueva tabla de páginas mutuamente excluyente para el proceso A.
3. Asignar memoria para el segmento de código (text) del proceso y copiar el contenido del fichero en memoria. Establecer los permisos de acceso.
4. Asignar memoria para el segmento de datos y copiar la sección de datos con valor inicial del archivo en la memoria. Establecer los permisos de acceso (lectura y escritura).
5. Asignar memoria para el segmento de bss (datos sin valor inicial) y llenarla de ceros. Establecer los permisos de acceso (lectura y escritura).
6. Asignar memoria para el heap del proceso y establecer los permisos de lectura y escritura.
7. Asignar memoria para la pila del proceso y establecer los permisos de lectura y escritura.
8. Copiar todas las bibliotecas compartidas necesarias y mapearlas en la tabla de páginas del proceso.
9. Establecer el puntero de instrucción al comienzo del segmento de código del proceso.
10. Transferir el control al proceso A para que comience su ejecución.

- b) 1. El archivo ejecutable se mapea directamente en la memoria del proceso en lugar de ser leído y cargado por separado. Esto puede mejorar el rendimiento al reducir la cantidad de E/S de disco necesaria.
2. Los segmentos de código y datos compartidos entre diferentes procesos que ejecutan el mismo archivo se comparten en la memoria física, lo que reduce el consumo de memoria.
3. Las modificaciones realizadas en las áreas de memoria compartida se reflejan directamente en el archivo de disco, lo que puede tener implicaciones de seguridad y rendimiento.
4. El SO debe gestionar adecuadamente los permisos de acceso y protección de las áreas de memoria compartida para garantizar la integridad del archivo y la seguridad entre los procesos.
5. La técnica de ficheros proyectados en memoria puede simplificar la implementación de ciertas funcionalidades del sistema, como la carga de bibliotecas compartidas y el intercambio de datos entre procesos.