Problema 1 (4 puntos en total, 0.5 puntos por pregunta)

```
1  int main(int argc, char **argv)
2  {
3    struct stat st;
4
5    int i, fd, len, countA, countB;
6    char *file_memory;
7
8    stat(argv[1], &st);
9    len = st.st_size;
10   countA = 0;
11   countB = 0;
12
13   fd = open(argv[1], O_RDWR, S_IRUSR | S_IWUSR)
14
15   file_memory = mmap(0, len, PROT_READ | PROT_
16               MAP_SHARED, fd, 0);
17
18   close(fd);
19
20   for(i = 0; i < len; i++)
21     if (file_memory[i] == 'a' || file_memor  i   = '/  )
22       countA++;
23
24   printf("Number of vowels (a/A) in the fi   :   l \r    countA);
25
26 // munmap(file_memory, len);
27
28   for (i = 0; i < len; i++)
29     if (file_me  ry    ]    'B')
30       countB++;
31
32   printf ("Number of vowels (b/B) in the fi   %d \n", countB);
33
34  munmap(file_memory, len);
35   return 0;
36 }
```

Al ejecutar el código anterior (./mmap file.txt) se uestran los siguientes mensajes por pantalla.

```
Number of vowels (a/A) in the file: 520
Number of vowels (b/B) in the file: 90
```

a) Comenta brevemente qué es lo que hace la línea 15 del código.

La línia 15 del codi el que fa és crear un arxiu mapat a memòria amb el fitxer que hem obert i del que tenim el *file descriptor*. El primer paràmetre és l'adreça, el segon el nombre de *bytes* que es volen mapar, el tercer és per indicar el tipus d'accés que volem permetre (lectura i escriptura en aquest cas), el quart són els flags (*MAP_SHARED* s'utilitza per compartir el mapat amb tots els altres processos que estan mapats a aquest objecte. Si hi ha canvis a la regió mapejada, s'escriuran al fitxer), el cinquè és el *file descriptor* del fitxer que volem mapar i l'últim és l'*offset*.

b) Al realizar el mmap, ¿se carga todo el fichero a memoria? Razona tu respuesta.

En el nostre cas si que es carrega tot el fitxer a memòria ja que la porció del fitxer que s'està mapejant comença des de l'*offset* (aquí 0) fins una llargada de *len bytes*. Fixem-nos que a la variable *len* l'hem assignat el valor del tamany total de l'arxiu representat en *bytes* gràcies a la funció *stat* i la seva variable *st.st_size*.

c) ¿Cuáles son las ventajas de mmap en el código con respecto a utilizar las llamadas a sistemas read/write?

Llegir i escriure d'un fitxer mapejat a memòria evita la còpia aliena que es produeix quan s'utilitzen les crides a sistema de lectura o escriptura, on les dades s'han de copiar a i des d'un *buffer* d'espai d'usuari.

A part de possibles errors de la pàgina, la lectura i escriptura d'un fitxer assignat a memòria no comporta cap sobrecàrrega de crida al sistema o canvi de context. És tan senzill com accedir a la memòria.

Quan diversos processos mapegen el mateix objecte a memòria, les dades es comparteixen entre tots els processos. Això pot estalviar molta memòria.

d) ¿Cómo es posible que podamos acceder al contenido del fichero abierto con identificador fd (línea 13) después de realizar un close del mismo (línea 18)? Razona tu respuesta.

Això és possible degut a que tancar el descriptor de fitxer no desmapeja la regió. La funció *mmap* afegeix una referència extra al fitxer associat amb el descriptor de fitxer que l'hem passat per paràmetre i aquest no s'elimina amb una crida a la funció *close*. El fitxer es manté referenciat mentre existeixi un mapat sobre el fitxer.

e) En las líneas 20-22 y 28-30 se contabiliza el número de ciertas vocales de un fichero. Si se realizaran cambios en el código y se reemplazan unas vocales por otras (As por Bs), ¿cada vez que se realiza un cambio en una vocal se guarda dicha modificación a disco? Razona tu respuesta.

La funció *mmap* no ens asegura que s'actualitzi a l'instant aquesta modificació al disc. Per assegurar-nos de que això succeeix hauriem de sincronitzar-la amb ajudes d'altres funcions. O bé esperar a que es cridés a *munmap*. Això és degut al flag *MAP_SHARED*.

f) ¿Qué es lo que hace la línea 34 del código (munmap)? ¿Por qué es necesario ejecutar esta línea? Es decir, ¿cuál es el objetivo de ejecutar esta línea?

La funció *munmap* elimina les assignacions per a l'interval d'adreces especificat i fa que les referències a adreces dins d'aquest interval generin referències de memòria no vàlides. La regió també es desmapeja automàticament quan finalitza aquest procés.

Si no s'executa aquesta línia però el nostre programa acaba amb un *exit*, no succeeix res ja que es cridarà automàticament a *munmap*. Tot i així, podem voler desfer el mapat a memòria per alliberar els recursos si els volem fer servir per un altre funció.

g) Si descomentamos la línea 26, ¿será posible acceder al fichero en las líneas 28-30

para contar las vocales b/B? Razona tu respuesta.

No serà possible. Bàsicament degut a que a l'inici del programa teniem dos referències a aquest fitxer (el seu *file descriptor* i la referència generada per la funció *mmap*). En aquest punt del programa ja hem fet un *close* (tancant el descriptor de fitxer) i, si a més desmapem de memòria el fitxer (amb *munmap*) ja no podrem accedir a aquest i el programa ens retornarà un *SIGSEGV* (senyal enviat a un procés quan es fa una referència a memòria no vàlida, o a una fallada de segmentació).

h) Si modificamos el código y reservamos memoria dinámica con un malloc, ¿seria posible escribir en esta parte de la memoria después de realizar un free? Razona tu respuesta.

Quan fem un *malloc* de memòria, el sistema ens retorna un punter a un espai específic. Quan fem un *free*, estem retornant aquest punter al sistema. A vegades després d'haver fet el *free* podem continuar escrivint a memòria, però no és una bona praxis ja que pot fer que sobreescrivim parts de memòria que no volíem.

Problema 2 (2.25 puntos en total, 0.75 puntos por pregunta) Supongamos que un sistema operativo con un solo procesador utiliza un algoritmo de planificación multi-level feedback queue (MFQ). En un instante determinado hay múltiples procesos en la lista de "ready" y sólo uno está ejecutándose. El usuario admin ejecuta un nuevo proceso y el sistema operativo debe decidir cómo realizar su planificación.

a) El MFQ tiene diversas colas con rebanadas de tiempo de 10ms, 20ms, 40ms y 80ms, cada una con prioridades diferentes, que corresponden a colas round robin. Describe en qué consiste la planificación round robin.

En la planificació round robin el planificador disposa d'una llista de tasques i les executa de forma ordenada, des de la primera fins a l'última, de manera circular.

Cadascuna de les tasques s'executen un temps màxim limitat, anomenat llesca de temps. Cada cop que finalitza la llesca el planificador fa un canvi de context a una altra tasca. La tasca també es pot executar menys temps.

En fer un canvi de context es posa la tasca actual al final de la llista de tasques (a l'estat de preparat) i s'agafa la primera tasca de la llista de tasques preparades (i es posa a executar).

b) Al comenzar a ejecutarse el proceso este empieza a leer datos de disco (que no están en el buffer interno del sistema operativo). ¿En qué nivel de la cola situará el sistema operativo al proceso? ¿En uno con una rebanada de tiempo baja (10ms) o alta (80ms)? Razona la respuesta.

Degut a que la lectura de dades de disc és un procés que es realitza en un temps relativament curt a la resta de processos amb càrrega computacional més alta, llavors s'assignarà una llesca de temps baixa (10ms).

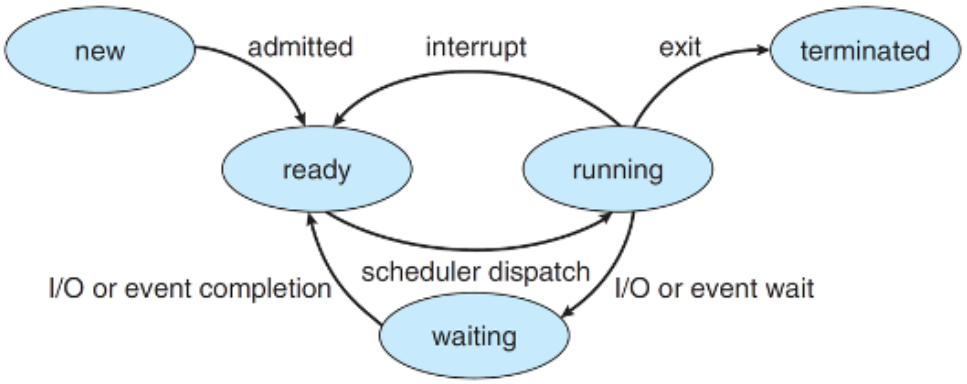
c) Una vez el proceso ha cargado los datos de disco comienza a realizar operaciones

matemáticas sobre ellas sin acceder a disco. ¿El sistema operativo moverá este proceso a otro nivel de la cola? Razonar la respuesta indicando, en su caso, a qué cola lo moverá el sistema operativo.

Les operacions matemàtiques es caracteritzen perquè tenen una càrrega computacional més elevada que un procés de lectura de disc, per tant el sistema operatiu s'encarregarà d'analitzar aquest cost i assignar segons quant temps podrà arribar a trigar. Com les operacions matemàtiques són computacionalment elevades i poden trigar molt més temps, s'assignarà una llesca de 80ms i ho mourà a l'última filera (prioritat més baixa).

Problema 3 (3.75 puntos en total, 0.75 puntos por pregunta) Comentar las siguientes afirmaciones de los sistemas operativos. En caso de que la frase sea incorrecta, indicar cómo funciona realmente y razonar la respuesta poniendo ejemplos si se cree conveniente. En caso que sea correcta, comentar la razón por la cual crees que es correcta.

a) Un proceso que se está ejecutando en un momento determinado puede pasar a la cola de "waiting" después de una interrupción.



Fals, després d'una interrupció el procés passa a l'estat de preparat (*ready*), degut a que el sistema prioritza la interrupció. Durant el procés associat a la rutina d'atenció a la interrupció el procés inicial segueix preparat (com a candidat a ser executat) esperant resposta de la interrupció i un cop aquesta finalitza, el procés inicial torna a executar-se en el mateix punt.

b) Las señales SIGUSR1 y SIGUSR2 pueden ser utilizadas para matar un proceso en ejecución.

Cert, de fet els senyals SIGUSR1 i SIGUSR2 estan "reservats" per a aplicacions d'usuari, perquè un procés pugui enviar una notificació a un altre procés, però si l'usuari no assigna cap acció, l'acció per defecte d'aquestes senyals és de terminar el procés, és a dir, matar-lo.

c) La lectura de un fichero grande (por ejemplo 4GB) mediante la llamada a sistema read es más efectiva (menor tiempo) que con la libreria de usuario fread.

Cert, per a fitxers grans amb la funció read es realitzarà una única crida a sistema per tal de realitzar la lectura del fitxer, en canvi amb fread es realitzaràn diverses crides per llegir el fitxer en blocs d'una mida determinada. El problema és que els blocs tenen una mida inferior a 4GB per tant es realitzaràn més de una crida a sistema, i degut al cost

computacional d'aquestes, és millor cridar una única vegada utilitzant read.
Per a fitxers petits, en canvi, és millor utilizar la funció fread.

d) En el momento de reservar memoria dinámica con un malloc se asignan las direcciones tanto en el espacio virtual como el físico.

Fals, aquesta funció només inicialitza el mapa de memòria virtual. L'assignació de marcs de pàgina físics es fa a mesura que hi accedim realment.

e) Las direcciones en el espacio físico y virtual se relacionan (mapean) mediante un marco de página

Fals, tot i que es pot realitzar el mapeig d'aquesta forma, no és l'única forma, també es pot relacionar mitjançant memòria segmentada, utilitzant diversos registres base i límit per a cada adreça virtual i així assignar parts de la memòria física a cadascuna.