Sistemas Operativos

Departamento de Computación – FCEyN – UBA
Segundo cuatrimestre de 2018

Corrigió Iván

Recuperatorio Segundo parcial – 04/12 - 2do. cuatrimestre de 2018

1	2	3	4	Nota
B	M	B	B	A

ACLARACIONES: 1) **Numere las hojas entregadas.** Esta hoja se entrega y es la hoja cero. Complete en la primera hoja la cantidad total de hojas entregadas (sin contar el enunciado). 2) Realice cada ejercicio en **hojas separadas** y escriba **nombre, apellido y L.U. en cada una.** 3) Cada ejercicio se califica con **Bien, Regular** o **Mal.** La división de los ejercicios en incisos es meramente orientativa. Los ejercicios se califican globalmente. El parcial se aprueba con 2 ejercicios bien y a lo sumo 1 mal/incompleto. 4) El parcial **NO** es a libro abierto. 5) **Justifique adecuadamente cada una de sus respuestas.**

Ejercicio 1.

Considere un archivo que consiste actualmente de 100 bloques. Suponga que la tabla de control de archivos (y la tabla con los índices, en el caso de la asignación indexada) ya está en la memoria. Calcule cuántas operaciones de E/S de disco se requieren para las estrategias de asignación contiguas, enlazadas e indexadas (de un solo nivel) si, para un bloque, se cumplen las siguientes condiciones. En el caso de asignación contigua, asuma que no hay espacio para crecer en el principio, pero hay espacio para crecer al final. También asuma que la información del bloque que se agregará se almacena en la memoria.

a) El bloque se añade al principio.

b) El bloque se agrega en el medio.

c) El bloque se añade al final.

d) El bloque se elimina desde el principio.

e) El bloque se elimina del medio.

f) Se quita el bloque del final.

Ejercicio 2.

El uso de polling para completar una E/S puede desperdiciar una gran cantidad de ciclos de CPU si el procesador itera un bucle de espera ocupado muchas veces antes de que se complete la E/S. Pero si el dispositivo de E/S está listo para el servicio, el polling puede ser mucho más eficiente que capturar y enviar una interrupción. Describa una estrategia híbrida que combine polling e interrupciones para el servicio de dispositivos de E/S. Para cada una de estas tres estrategias (polling puro, interrupciones puras, híbrido), describa un entorno informático en el que esa estrategia sea más eficiente que cualquiera de las otras.

Ejercicio 3.

Dado el siguiente código en lenguaje C:

```
int main (int argc, char * argv[]) {
  char buffer[64];
  int number = 50;
  if (argc != 2) {
    return -1;
  }
  strcpy(buffer, argv[1]);
  printf(buffer);
  printf("\n");
  printf(" ( -) Valor @ 0x %08x = %d 0x %08x\n", &number, number, number);
  return 0;
}
```

Describa cuál es la vulnerabilidad que posee, cuál es el ataque que suele utilizarse para explotarla, y qué medidas recomienda para evitarla.

Ejercicio 4.

Un sistema distribuido usa la siguiente interfaz para realizar el paso de mensajes:

- **send(i, msg)** – envía el mensaje en el buffer `msg` al proceso con identificador i.
- **receive(i, &msg)** – recibe un mensaje del proceso con identificador i, en el buffer `msg`.

El sistema distribuido está compuesto por N procesos, donde los identificadores de proceso i oscilan entre 0 y $N-1$, y donde cada proceso ejecuta el siguiente código. Las variables `msg` y `j` son locales y se desconoce su valor inicial.

```
01: mi_funcion(int i) {
02:   if (i == 0) {
03:     send(i+1, msg);
04:   }
05:   while(1) {
06:     if (i == 0) {
07:       j = N;
08:     } else {
09:       j = i;
10:     }
11:     receive(j-1, &msg)
12:     // Código del proceso i
13:     if (i == N-1) {
14:       j = -1;
15:     } else {
16:       j = i;
17:     }
18:     send(j+1, msg);
19:   }
20: }
```

Conteste razonadamente a las siguientes cuestiones:

a) ¿Cuántas veces como máximo puede ejecutarse concurrentemente la operación receive de la línea 11?
b) ¿Cuántas veces como máximo puede ejecutarse concurrentemente el código en la línea 12?
c) ¿Cómo están ordenados lógicamente los procesos que ejecutan en este sistema distribuido?
d) ¿Qué aplicación tiene este algoritmo?

1) Asignación Contigua:

a) Deben moverse los 100 bloques para liberar el primer espacio, y luego agregar el nuevo bloque [101]

b) Deben moverse los 50 bloques del final para liberar el del medio, y luego agregar el nuevo bloque [51]

c) Dado que sabemos donde termina el archivo, solo debemos agregar el bloque [1]

d) Eliminamos el bloque pero, dado que nuestro archivo sigue estando en bloques contiguos, no movemos el resto [1]

e) Eliminamos el bloque y recalculamos la segunda mitad [50]

f) Idem d [1]

Asignación enlazado:

Dado que agregar o remover bloques implica simplemente agregar el bloque al disco y, luego, modificar los enlaces, todas las operaciones implicarán una sola operación de E/S. Agregarlo al principio será, simplemente, modificar la tabla para que el nuevo bloque sea el primero del archivo, y que el anterior primero sea el que lo sigue. En el medio o final, similar.

Y borrar un bloque implicará borrarlo del disco y, luego, modificar los enlaces de forma que ese bloque se considere vacío y el archivo no lo considere parte de él

Asignación indexado:

Al tener un solo nivel, no necesitamos [redacted] utilizar indirecciones. De esta forma, las operaciones implican una sola operación de E/S de disco.

Nuevamente, cargamos el bloque al disco y, luego, modificamos los punteros de los índices de forma que, al cargar el último índice, se obtenga el nuevo bloque. Lo mismo en la mitad, lo mismo en el final. Al eliminar un bloque el proceso es similar notar que tenemos el puntero a cada uno de los bloques del archivo, por lo que no debemos cargarlo sino simplemente modificar el índice que tiene en nuestra estructura

N° 1 FECHA

2) Un ejemplo de estrategia híbrida sería utilizar interrupciones a la hora de recibir el fallo del dispositivo de E/S, y luego usar polling durante la espera de que ese proceso finalice. Esta estrategia sería útil en el caso de un dispositivo que no suele ser utilizado, pero que al lanzar un proceso lleva al CPU a realizar una serie de tareas que deben ser atendidas con urgencia. De esta forma, la interrupción inicial nos permite atender inmediatamente este dispositivo, y el polling evita que, al acabar el mismo, se interrumpan el resto de los procesos lanzados.

X

Polling: Es útil, por ejemplo, al utilizar sensores de temperatura. Se espera que se renueve constantemente el valor, pero no que esto interrumpa el resto de los procesos

Interrupciones: Un caso sería una alarma. Esta [redacted] debe ser atendida con urgencia, pero no este valor que ocurre con frecuencia, por lo que usar polling sería un desperdicio de CPU. [redacted]

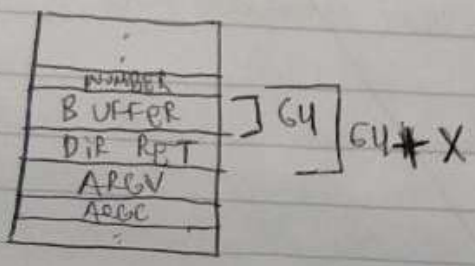
Híbrido: Caso de impresora, en la cual debemos considerar la posibilidad de que se quede sin tinta y que termine de imprimir. Para dar aviso del comienzo y final de la impresión usamos interrupciones, pero para chequear que tenga tinta durante la misma utilizamos polling, y solo debido a una serie de intentos fallidos damos aviso al usuario de que no tiene tinta.

Faltan algunas hipótesis p/ el ejemplo de la impresora.

Mal la justificación del enfoque híbrido.

NOTA

3) La vulnerabilidad que posee es que, al reservar 64 bytes de buffer, podría ocurrir que un usuario [redacted] lote un argumento de mayor longitud y, al correr la instrucción Strcpy se pisen elementos del stack. Un ataque para explotar esta vulnerabilidad es, efectivamente, pasar un valor en argv[1] que pise los registros y la dirección de retorno. De esa manera, al finalizar la función la línea tendrá un dir ret diferente, pudiendo ejecutarse código malicioso. A través de este ataque, un usuario sin los permisos correspondientes podría crear un código que, al ser llamado desde una función con privilegios mayores, obtenga información sensible, la modifique o la borre.



Strcpy copia todo el string, entonces si toma ARGV[1] de tamaño 64+X, pisará el valor.

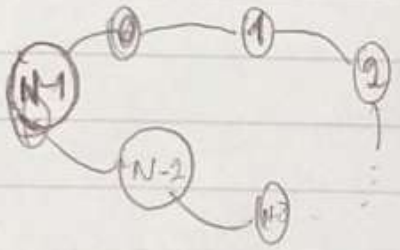
Para evitar estos ataques, podría utilizarse stack-randomization, que antes de correr el proceso coloca el stack en una posición random de memoria, de forma que el usuario malicioso no sea capaz de saber en qué posición de memoria se encuentra el código que él desea correr. Sin embargo, sigue siendo una solución parcial, pues aún así la dir ret es modificable y, si bien es mínimo, existe la posibilidad de un ataque exitoso.
Una mejor solución sería utilizar un stack-canary; un valor aleatorio que se coloca en el stack por encima de la dir ret y, al finalizar la ejecución, se chequea si fue modificado. De ser así, se detiene la ejecución. Dado que el usuario malicioso debe, indefectiblemente, pisar este valor para poder cambiar la dir ret, el ataque es anulado.

Asumo send y receive operaciones bloqueantes

4) a) Por la forma en la que está implementado el algoritmo, podría ocurrir que N (N-1 procesos) se encuentren ejecutando concurrentemente la operación receive. Dado que cada proceso espera el mensaje del proceso anterior (o, en el caso del proceso 0, del n-1 ésimo) y que solo el proceso 0 tiene un send previo al receive, los procesos desde el 1 al N-1 podrían ejecutar concurrentemente este código hasta que todos hayan recibido los saludos correspondientes (el 0, de que el 1 recibió; el 1, de que el 0 mandó). Notemos que durante toda la ejecución tendremos, como máximo, un proceso haciendo un send y N-1 esperando un receive (o en camino a hacerlo).

b) Siguiendo la lógica del punto anterior, solo un proceso ejecutará el código de la línea 11 de forma concurrente, pues será el que recibió el mensaje del proceso i-1 y, llegada la línea 18, le enviará un mensaje a i+1.

c) Los procesos están ordenados de forma circular de manera que cada proceso se comunica con i-1 y con i+1 (en el caso del 0, con N-1 y con 1). Notemos que, desde que se lanzan los N procesos, se siguen enviando mensajes consecutivamente hasta que alguno de ellos deje de hacerlo.



(el tamaño es producto de un mal dibujo, y si, eso es un círculo)

d) Podría usarse para asegurarse que los N procesos sigan vivos. El mensaje podría ser un timestamp, y que el mismo refiriera "la última vez que los N procesos dijeron que estaban vivos", en la primera vuelta no tendría sentido, pues podría darse el envío en el 7mo receive y que los primeros 6 tengan una falsa información. De esta manera, el N-1 ésimo proceso sería el encargado de detectar el mensaje, y en ese caso cada vez que ese proceso reciba un mensaje le avisará al 0 que se dio una vuelta completa en ese instante.

Ok, pero más interesante es hablar de token ring y la sección crítica.