Sistemas Operativos - Examen 1

June 1, 2020

{Prof:jgonzalez}@utec.edu.pe

Instrucciones:

- Leer los enunciados y **dar respuestas legibles de acuerdo a lo requerido** en cada ejercicio.
- **Justificar** todas su respuesta de forma **clara y directa**.
- Desarrollar el examen **solamente con los miembros de su grupo de PintOS**.
- La resolución presentada debe ser a mano, letra clara y legible.
- **Es permitido** usar material escrito o electrónico para desarrollar el examen. **Observación:** no es permitido copiar y/o traducir segmentos de libros o material online, interpretar y resolver según requiere el ejercicio.
- **Es permitido usar lápiz**, siempre y cuando la respuesta sea legible.
- Presentar sus respuestas en un solo archivo Examen1.zip el cual contiene: a) Solucion.pdf (incluir nombres y códigoID de los estudiantes, fotos legibles o scan de la resolución).

1. Responder:
   (a) (½ point) Por qué la memoria principal no es apropiada para almacenar de forma permanente los programas o para almacenar backups? Está relacionado al protocolo de coherencia por directorios?
   (b) (1 point) Definir y explicar el propósito de un *interrupt vector*.
   (c) (1 point) Existen dos formas en que los comandos son procesados por un command interpreter. Una es permitir que el command interpreter contenga el código necesarioo para ejecutar el comando. la otra es implementar los comandos a través de system programs. Comparar y contrastar ambos approaches.
   (d) (1 point) Indicar cuales son las ventajas o desventajas de usar un approach microokernel.
   (e) (½ point) Algunos sistemas UNIX tienen dos versiones de fork(). Describir la función de cada uno así como decidir cual usar.
   (f) (½ point) Cual es la diferencia entre un semaphore y un conditional variable?
   (g) (1 point) Indicar un escenario (cuantitativo) en el problema de filósofos usando monitores, en el cual un filósofo *starve to death*.
   (h) (1 point (bonus)) Describir los mecanismos usados para synchronization en el Kernel de Windows en sistemas single y multiprocessor. Explicar porque Windows usa diferentes mecanismos para estos dos sistemas.

2. (2½ points) Cinco filósofos llevan una vida monótona alrededor de una mesa: estos piensan, les da hambre y comen. Considere que en un día de fiesta, fueron autorizados a tomar vino. Como ellos están acostumbrados a compartir recurso, apenas tres (3) copas fueron colocadas en el centro de la mesa. Animados con la novedad, los filósofos F0, F1, F2 decidieron primero disputar la copa y despues los tenedores con los vecinos. Los filósofos F3 y F4 decidieron primero disputar los tenedores y después la copa. Este algoritmo basado en semáforos está sujeto a deadlock? Justifique su respuesta.

```
sem_t tenedor[5]={1,1,1,1,1};
sem_t copa=3;
```

F0,F1,F2:

```
while (1){
        piensa();
        sem_wait(&copa);
        sem_wait(&tenedor[i]);
        sem_wait(&tenedor[(i+1)%N]);
        come();
        sem_post(&copa);
        sem_post(&tenedor[i]);
        sem_post(&tenedor[(i+1)%N]);
}
```

F3, F4:

```
while (1){
        piensa();
        sem_wait(&tenedor[i]);
        sem_wait(&tenedor[(i+1)%N]);
        sem_wait(&copa);
        come();
        sem_post(&copa);
        sem_post(&tenedor[i]);
        sem_post(&tenedor[(i+1)%N]);
}
```

3. (4 points) Demostrar el correctness del algoritmo de Dekker:

```
boolean flag [2];
int turn;
void P0()
{
  while (true) {
     flag [0] = true;
     while (flag [1]) {
        if (turn == 1)
          flag [0] = false;
              while (turn == 1) /* do nothing */;
              flag [0] = true;
              }
    }
    /* critical section */; turn = 1;
    flag [0] = false;
    /* remainder */;
    }
}
void P1( ) {
    while (true) {
      flag [1] = true;
      while (flag [0]) {
          if (turn == 0) {
              flag [1] = false;
              while (turn == 0) /* do nothing */;
              flag [1] = true;
```

```
                }
            }
        /* critical section */;
        turn = 0;
        flag [1] = false;
        /* remainder */;
        }
}
void main ()
{
  flag [0] = false;
  flag [1] = false;
  turn = 1;
  parbegin (P0, P1);
}
```

(a) Mostrar que la mutual exclusion es enforced. TIP: mostrar que cuando P_i entra a la sección crítica, la siguiente expresión es *true*:

```
flag[i] and (not flag[1-i])
```

(b) Mostrar que un proceso solicitando acceso a la sección crítica no va a ser atrasado indefinidamente. TIP: considerar los siguientes casos: (1) un single process trata de acceder a la sección crítical, (2) ambos procesos intentan acceder a la sección crítica, y (2a) `turn=0` y `flag[0]=false`, y (2b) `turn=0` y `flag[0]=true`

(c) Considerar el algoritmo de Dekker escrito para un número arbitrario de procesos cambiando la expresión ejecutada al salir de la sección crítica de:

```
turn = 1 - i //i.e. P0 sets turn to 1 and P1 sets turn to 0
```

hacia:

```
turn = (turn + 1) % n /* n = number of processes */
```

Evaluar el algoritmo cuando el número de procesos ejecutándose concurrentemente es mayor a dos (2).

4. (4 points) El siguiente pseudocódigo es la correcta implementación del producer/consumeer problem con un bounded buffer:

   Los labels p1,p2,p3 y c1,c2,c3 se refieren a las líneas de código mostrado arriba (p2 y c2 cubren tres líneas de código). Los semaphores empty y full son semaphores lineales que pueden tomar valores unbounded positivos y negativos. Existen múltiples procesos productores referidos como Pa, Pb, Pc, etc. y múltiplos procesos consumidores, referidos como Ca, Cb, Cc, etc. Cada semaphore mantiene un queue FIFO de blocked processes. En la tabla de abajo de scheduling, cada línea representa el state del buffer y semaphores después que la scheduled execution ha sucedido. Para simplificar, asumimos que el scheduling es tal que los procesos nunca son interrumpidos mientras ejecutan ua porción de código dada p1, o p2, ..., o c3.

   (a) Completar la tabla. Mostrar su solución.

5. (4 points) Cinco batch jobs, A hasta E, llegan a un cluster esencialmente al mismo tiempo. Estos tienen un running time de 15,9,3,6 y 12 minutos respectivamente. Sus prioridades (externamente definidas) son 6,3,7,9 y 4 respectivamente, con el valor más bajo correspondiente a la más alta prioridad. Para cada uno de los siguientes algoritmos de scheduling, determinar el *turnaround* time para cada proceso y el average *turnaround* para todos los jobs. Ignorar el process switching overhead. Explicar sus soluciones. En los últimos tres casos, asumir que solo un trabajo por vez ejecuta hasta que finaliza, y todos los trabajos son completamente processor bound.

```
item[3] buffer; // initially empty
semaphore empty; // initialized to +3
semaphore full; // initialized to 0
binary_semaphore mutex; // initialized to 1
```

```
void producer()
{
     ...
     while (true) {
        item = produce();
p1:     wait(empty);
  /     wait(mutex);
p2:     append(item);
  \     signal(mutex);
p3:     signal(full);
     }
}
```

```
void consumer()
{
     ...
     while (true) {
c1:     wait(full);
  /     wait(mutex);
c2:     item = take();
  \     signal(mutex);
c3:     signal(empty);
        consume(item);
     }
}
```

Scheduled Step of Execution	**full's State and Queue**	**Buffer**	**empty's State and Queue**
Initialization	full = 0	OOO	empty = +3
Ca executes c1	full = −1 (Ca)	OOO	empty = +3
Cb executes c1	full = −2 (Ca, Cb)	OOO	empty = +3
Pa executes p1	full = −2 (Ca, Cb)	OOO	empty = +2
Pa executes p2	full = −2 (Ca, Cb)	X OO	empty = +2
Pa executes p3	full = −1 (Cb) Ca	X OO	empty = +2
Ca executes c2	full = −1 (Cb)	OOO	empty = +2
Ca executes c3	full = −1 (Cb)	OOO	empty = +3

(a) round robin con time quantum de 1 minuto.

(b) priority scheduling

(c) FCFS (ejecuta en orden 16, 9, 3, 6, y 12)

(d) shortest job first

Question:	1	2	3	4	5	Total
Points:	5½	2½	4	4	4	20
Bonus Points:	1	0	0	0	0	1
Score:						

Scheduled Step of Execution	full's State and Queue	Buffer	empty's State and Queue
Pb executes p1	full =		empty =
Pa executes p1	full =		empty =
Pa executes __	full =		empty =
Pb executes __	full =		empty =
Pb executes __	full =		empty =
Pc executes p1	full =		empty =
Cb executes __	full =		empty =
Pc executes __	full =		empty =
Cb executes __	full =		empty =
Pa executes __	full =		empty =
Pb executes p1-p3	full =		empty =
Pc executes __	full =		empty =
Pa executes p1	full =		empty =
Pd executes p1	full =		empty =
Ca executes c1-c3	full =		empty =
Pa executes __	full =		empty =
Cc executes c1-c2	full =		empty =
Pa executes __	full =		empty =
Cc executes c3	full =		empty =
Pd executes p2-p3	full =		empty =