PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC 2333 — Sistemas Operativos y Redes
Interrogación 1

Miércoles 07-Abril-2015

Duración: 2 horas

1. **[10p]** Indique si las siguientes afirmaciones son verdaderas o falsas. En caso que sean falsas, debe explicar por qué lo son, de lo contrario no tendrán puntaje. (1 cada una)

   1.1) Una situación de *deadlock* puede ser resuelta utilizando un algoritmo de *scheduling* expropiativo.

   1.2) Un sistema operativo que intenta otorgar una buena respuesta a la interacción del usuario con el escritorio, debe ser capaz de privilegiar los procesos intensivos en E/S.

   1.3) Un sistema operativo puede funcionar sin interrupciones por *timer*

   1.4) El *stack pointer* (SP) de un proceso sólo puede ser modificado en modo monitor.

   1.5) Es posible tener un sistema con multiprogramación, pero sin *multi-tasking*.

   1.6) Sistemas operativos como Windows o Linux evitan *deadlock*s utilizando el algoritmo del banquero.

   1.7) En un *scheduler Round-Robin*, mientras más pequeño sea el *quantum* mayor será la frecuencia a la cual los procesos son asignados a la CPU.

   1.8) Un programa, originalmente secuencial, ha sido modificado de manera que el $95\,\%$ de sus instrucciones pueden ser ejecutadas en paralelo. Al ejecutarlo en un cluster con miles de *cores*, la ejecución toma un 1/20 del tiempo original.

   1.9) Cada vez que un proceso hace una llamada al sistema, se ejecuta un cambio de contexto (i.e., se llama al *scheduler*)

   1.10) Un sistema que se encuentra en un estado *inseguro*, en el futuro se encontrará en *deadlock*

2. **[10p]** Responda de manera breve y precisa las siguientes preguntas:

   2.1) **[4p]** Suponga que tiene una línea de comandos con 3 operaciones: `head -n X file`, que retorna las primeras `X` líneas del archivo `file`; `tail -n X file`, que retorna las últimas `X` líneas de `file`; y `grep patron file` que retorna solamente las líneas de `file` que contienen el *substring* `patron`. En todos los casos, si se omite `file`, el comando lee desde la entrada estándar (`stdin`).

   Describa el resultado de las operaciones[1]:

   a) `head -n 100 thread.c | tail -n 40 | grep ''struct thread''`

   b) `grep ''ticks'' thread.c | head -n 20 | tail -n 1`

   2.2) **[3p]** Un *lock* implementado usando espera ocupada se conoce como *spinlock*. Se argumenta que los *spinlock*s pueden ser convenientes para ser utilizados en multiprocesadores. Justifique si esto es una buena idea o no.

   2.3) **[3p]** Para el siguiente código, considera los siguientes prototipos:

   - `pid_t fork()` retorna 0 en el caso del hijo y el *pid* del hijo, en el caso del padre.
   - `int exec(command)` recibe como parámetro la ruta del archivo con el código a ejecutar
   - `pid_t wait(*exitStatus)` recibe como parámetro un puntero donde se guarda el estado de salida del proceso que ha terminado (puede usar `NULL` si no le interesa ese valor). Retorna el *pid* del proceso que terminó, ó -1 si el proceso no ha hecho `fork()`

[1] Ejemplo: `head -n 100 thread.c` muestra las primeras 100 lineas de `thread.c`

Para el siguiente código escriba su salida en pantalla. Puede haber muchas salidas válidas. Escriba cualquiera de ellas, pero sólo una.

```c
int main(int argc, char *argv[]) {
  pid_t pid;
  int a = 42;
  printf("Wow!\n");
  pid = fork();
  if(pid == 0) {
    pid = fork();
    a++;
    if(pid == 0) {
      printf("3(%d)\n",a);
    }
    else {
      pid = wait(NULL);
      if(pid > 0) {
        printf("4(%d)\n",a);
      }
    }
  }
  else {
    pid = fork();
    if(pid == 0) {
      a++;
      printf("1(%d)\n",a);
      execl("/bin/date", "date", NULL);
      printf("5\n");
    }
    printf("2(%d)\n",a);
  }
  printf("0\n");
  return 0;

}
```

3. **[14p]** Considere un tramo de la Ruta 5 Norte con dos pistas, una en cada sentido. Un tramo de ella, por razones climáticas, se ha dañado de manera que en ese tramo solo es posible circular en un sentido a la vez. La longitud del tramo dañado permite que haya un máximo de N vehículos circulando en él.

   3.1) **[7p]** Escriba un código para `vehiculoNorte` y uno para `vehiculoSur` de manera que en todo momento solo haya un tipo de vehiculo circulando por ese tramo de la carretera. Los vehículos que no están circulando deben esperar a que pasen todos los del otro sentido antes de intentar entrar. Especifique las variables compartidas y su inicialización.

   3.2) **[7p]** Modifique su solución para que, después que un máximo de M vehículos ($M < N$) vehiculos pasen en un sentido, se le de la oportunidad a los del sentido opuesto. Esta modificación le agrega justicia (espera acotada) al problema.

   La solución puede utilizar las primitivas de software vistas en clase: *locks*, semáforos, monitores/variables de condición, pero no *busy-waiting*. También puede implementar un solo tipo de proceso `vehiculo` que sea capaz de comportarse correctamente en cada sentido.

4. **[16p]** Considere un conjunto de procesos $P = \{p_1, p_2, p_3, p_4\}$ con ráfagas (*burst-time*) $T = \{10, 6, 31, 9\}$, tiempos de llegada $A = \{0, 4, 8, 9\}$. Determine el tiempo de espera (*waiting time*) promedio, y la cantidad de cambios de contexto generados por los siguientes algoritmos de *scheduling* (los tiempos están en *ms*):

4.1) **[4p]** SJF sin expropiación (*non-preemptive*)

4.2) **[4p]** Round Robin, con $q = 8$

4.3) **[4p]** Prioritario con expropiación, con prioridades $Pr = \{5, 40, 30, 50\}$

4.4) **[4p]** Round Robin en que $q = 8$, y cada vez que un proceso utiliza todo su *quantum*, para el próximo turno su *quantum* se duplica.

Cuente los cambios de contexto aunque el proceso elegido sea el mismo que estaba ejecutando. Incluya el cambio de contexto en el instante 0.

5. **[10p]** Considere un pub que cuenta con 7 mesas, 14 sillas, y 14 vasos para sus clientes. Tres grupos de amigos (G_1, G_2, G_3) se encuentran en el pub. Al llegar, cada grupo declaró cuantos recursos ocuparían. G_1 declaró que ocuparía 2 mesas, 6 sillas y 7 vasos; G_2 declaró que ocuparía 5 mesas, 8 sillas y 7 vasos; y G_3 declaró que ocuparía 3 mesas, 7 sillas y 10 vasos. Sin embargo, como no todos los miembros del grupo han llegado, el pub les ha dado solamente algunos de los recursos que necesitan y les asignará el resto a medida que lleguen los miembros faltantes.

   Hasta el momento, G_1 está ocupando 1 mesa, 4 sillas y 7 vasos; G_2 está ocupando 2 mesas, 2 sillas, y 5 vasos; y G_3 está ocupando 2 mesas, 6 sillas, y 1 vaso. Ningún grupo se retirará mientras no haya utilizado todos los recursos que declaró, pero es posible dejarlos en espera (el mesero puede decidir ignorarlos temporalmente). En cualquier caso, se requiere que todos los grupos puedan ser atendidos completamente en algún momento y que ninguno se quede esperando hasta el cierre del pub.

   - **[4p]** Dada esta situación, ¿es posible satisfacer las demandas restantes de todos los grupos? Justifique por qué si o por qué no.
   - **[3p]** A partir de la situación inicial, si G_2 solicita dos mesas más, ¿debe el mesero atender su solicitud?
   - **[3p]** A partir de la situación inicial, si G_3 solicita 1 mesa y 1 un vaso más, ¿debe el mesero atender su solicitud?

6. **[5p] Esta pregunta es OPCIONAL**. Describa de la manera más breve y precisa posible, el mecanismo que utiliza Pintos para efectuar los cambios de contexto (lo que ocurre desde que se llama a `schedule`).

   No responda esta pregunta si no está seguro o no recuerda nada del código.