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

IIC2333 — Sistemas Operativos y Redes — 1/2017

Interrogación 1

Viernes 28-Abril-2017

Duración: 2 horas
SIN CALCULADORA

1. **[12p]** Consider el siguiente pseudocódigo (C-like) para solucionar el problema de la sección crítica entre n *threads* ejecutando en una única CPU. Las únicas instrucciones atómicas son las comparaciones y asignaciones individuales.

```
// variables compartidas
bool escogiendo[n] = {FALSE, FALSE, ... , FALSE};
int numero[n] = {0, 0, ..., 0};
```

El siguiente código es ejecutado por cada *thread* i.

```
1   do {
2     escogiendo[i] = TRUE;
3     numero[i] = max(numero[0], numero[1], ..., numero[n-1]) + 1;
4     escogiendo[i] = FALSE;
5     for (j=0; j<n; j++) {
6       while(escogiendo[j] == FALSE);
7       while(numero[j] > 0 && (numero[j] < numero[i]) );
8     }
9     /** SECCION CRITICA **/
10    numero[i] = 0;
11    /** resto del codigo **/
12  } while(true);
```

1.1) **[9p]** Argumente si esta propuesta cumple o no con cada una de las condiciones de solución al problema de la sección crítica.

1.2) **[3p]** Indique cómo mejorar esta solución para que cumpla con todas las condiciones del problema de la sección crítica. No es necesario argumentar por las condiciones que se siguen cumpliendo.

2. **[7p]** El siguiente código intenta ser replicar un comando `N` veces **en paralelo**.

```
char *command;
char *arguments;
// puede agregar mas variables aqui
while('r') {
  read_command(&command, &arguments); // lee correctamente el comando y sus argumentos
  /** completar codigo aqui **/

}
```

Complete este código, de manera que luego de ejecutar los `N` procesos en **paralelo**, el programa escriba en pantalla cuántos terminaron correctamente (esto es, su *exit code* es 0), y cuántos no.

3. **[11p]** Deadlock y *scheduling*

   - **[3p]** El algoritmo del banquero tiene como premisa mantener al sistema en un estado seguro para evitar *deadlocks*. Sin embargo, un estado inseguro no es necesariamente un estado de *deadlock*. ¿Por qué se desea mantener al sistema en un estado seguro?
   - **[3p]** ¿Por qué un *scheduler* FCFS (*First-Come First-Serve)* no es apropiado para procesos interactivos? Mencione uno que sí lo sea, y justifique brevemente por qué.
   - **[5p]** De los siguientes elementos, ¿cuáles deben ir en un PCB, y cuáles no? (solo mencionarlo): variables locales de funciones, ubicación de la tabla de páginas, prioridad de *scheduling*, código binario del proceso, ubicación de TLB.

4. **[30p]** Considere un sistema computacional para soportar hasta 1024 procesos concurrentes, todos con un comportamiento similar. El *hardware* está diseñado para soportar hasta 8GB de memoria física, y el sistema operativo configurado para manejar páginas de 16KB. El espacio de direcciones virtuales utiliza 46 bit.

   4.1) **[7p]** Diseñe un sistema de direccionamiento de memoria usando tabla de páginas de un nivel. Especifique, describiendo claramente su cálculo, la cantidad de bit para *offset*, número de página, número de *frame*, tamaño en memoria de la tabla de páginas, y cantidad total de memoria virtual direccionable.

   Incluya al menos 3 bit de metadata, indicando cuáles son.

   Considere además que la arquitectura requiere que cada entrada en la tabla de páginas (PTE) debe ser de un tamaño que sea múltiplo de 1 Byte (8 bit). Esto significa que puede necesitar agregar bits adicionales (*padding* o alineamiento).

   4.2) **[7p]** Modifique el diseño de la tabla de páginas utilizando paginación multinivel, de manera que cada tabla en cada nivel quepa **completamente en una página**.

   Especifique la cantidad de bit y los tamaños de tabla en cada nivel, e indique la cantidad mínima de memoria física necesaria para poder hacer una traducción.

   4.3) **[4p]** ¿Cuánto espacio ocuparía en memoria utilizar una tabla de páginas invertida?

   4.4) **[3p]** De acuerdo a los bit que especificó en la primera parte, ¿qué algoritmo de reemplazo de páginas utilizaría, y por qué?

   4.5) **[3p]** Suponga que el $90\,\%$ de los accesos a memoria corresponde a un conjunto de 10 páginas. En el esquema de una tabla con un nivel, ¿de qué tamaño (en Byte) debería ser el TLB para tener un $90\,\%$ de *hit rate*? Justifique su respuesta. Recuerde que cada entrada debe ser del tamaño de un múltiplo de 1 Byte.

   4.6) **[3p]** Suponiendo que una consulta al TLB toma Ans, y un acceso a memoria toma Bns, ¿cuánto sería el tiempo de acceso promedio a memoria usando el TLB de la pregunta anterior?

   4.7) **[3p]** ¿Cuál debería ser un criterio para el *long term scheduler* al momento de decidir si permite aceptar un proceso para ejecución?

API de procesos

- `pid_t fork()` retorna 0, en el contexto del hijo; retorn *pid* del hijo, en el contexto del padre.
- `int exec(char *command, char *argumentos)` recibe como parámetro un *string* con la ruta del archivo a ejecutar y sus argumentos. Si hay error retorna -1. De lo contario, no retorna.
- `pid_t wait(pid_t p, int *exitStatus)` espera por el proceso `p`, y guarda el estado de salida de `p` en `exitStatus`. Si `p` es -1, espera por cualquiera. Retorna el `pid` del proceso que hizo `exit`.

i	2^i
6	64
7	128
8	256
9	512
10	1024

i	2^i
10	1 KB
20	1 MB
30	1 GB
40	1 TB