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

IIC 2333 — Sistemas Operativos y Redes — 2/2016

Interrogación 1

Lunes 29-Agosto-2016

Duración: 2 horas
Una HOJA por pregunta. Para cada pregunta se corregirá **máximo una hoja por pregunta**. (4 hojas en total)

1. **[8p]** Considere un sistema operativo con un espacio de direcciones de 16-bit. Todos los procesos que son cargados en memoria reciben un espacio de 4 KB y se cargan secuencialmente. Si no hay espacio en memoria, no es posible cargar un proceso. El sistema operativo ejecuta los procesos en el orden que son cargados en memoria. Si un proceso se bloquea en una operación de I/O o si termina, el sistema ejecuta el siguiente proceso. Una vez que la operación de I/O termina, el proceso puede ser nuevamente elegido para ejecutar. Un proceso que termina libera su espacio en memoria para que otro se cargue en él. De acuerdo a esta descripción, justifique si este sistema posee *multiprogramación* y/o *multitasking*.

2. **[15p]** Para las siguientes afirmaciones, indique si está de acuerdo con ella, y justifique su respuesta.

   2.1) *Multitasking* equivale a decir que múltiples usuarios pueden usar el computador concurrentemente.

   2.2) La instrucción para modificar el *kernel bit* debe poder ejecutarse en modo usuario, de lo contrario no sería posible ejecutar *syscall*s.

   2.3) Todas las *syscalls* retornan el control al usuario.

   2.4) Una interrupción y una *syscall* con sinónimos de maneras de llamar al sistema operativo.

   2.5) Cuando un proceso entra en estado *zombie*, es seguro borrar la memoria que tiene asignada.

3. **[12p]** Respoda breve y concisamente las siguientes preguntas:

   3.1) Para almacenar la información de ejecución de un proceso se utiliza el *Process Control Block* (PCB). ¿Por qué esta información no se almacena en el espacio de memoria que el sistema operativo otorga al proceso para su ejecución?

   3.2) Explique en qué consiste el mecanismo *copy-on-write*, y para qué se aplica.

   3.3) ¿Qué ventajas y desventajas tiene el modelo de *thread*s *many-to-one* respecto a *one-to-one* ?

4. **[25p]** Para las siguientes preguntas considere estas definiciones de *syscalls*:

   - `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 un *string* con la ruta del archivo a ejecutar y sus argumentos.
   - `pid_t waitpid(pid_t p, int *exitStatus)` espera por el proceso `p`, y guarda el estado de salida de `p` en `exitStatus`
   - `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`.
   - `sleep(int secs)`. Duerme a un proceso durante `secs` segundos.

4.1) **[10p]** Para el siguiente código, indique cuántos procesos se generan y dibuje un árbol de procesos apropiado.

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
    fork();
    fork() && fork() || fork();
    fork();

    printf("forked %d\n", getpid());
    return 0;
}
```

Recuerde que, en C, los operadores lógicos son asociativos por la izquierda; que el AND lógico tiene precedencia sobre el OR; y que se usa evaluación de cortocircuito (*short-circuit evaluation*).

4.2) **[10p]** Escriba un código en C que emule una *shell* (línea de comandos). Esto es, el programa debe esperar un comando del usuario, crear un proceso que ejecute el comando, esperar su término, y luego volver a esperar un comando. Suponga una función `readline(char *cmd)` que lee un comando y lo guarda en `cmd`. No considere los *headers* de C.

4.3) **[5p]** Escriba un código en C que genere **exactamente** 1 proceso zombie. No considere los *headers* de C.