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

IIC2333 — Sistemas Operativos y Redes — 2/2017

Interrogación 1

Lunes 29-Agosto-2017

Duración: 2 horas
SIN CALCULADORA

1. **[21p]** Responda **brevemente** las siguientes preguntas

   1.1) **[5p]** De las siguientes operaciones, indique cuáles deben ser ejecutados en modo *kernel*, y cuáles pueden ser ejecutadas en modo *user*. Justifique **brevemente**.

   a) Invocar una *syscall* desde un programa en C
   b) Copiar el espacio de direcciones de un proceso desde la RAM al disco
   c) Leer un mensaje recibido por la interfaz de red
   d) Establecer el valor del *timer* para que genere una interrupción
   e) Modificar el registro PC (*Program Counter*)

   1.2) **[2p]** En los años 1960s se introdujo la multiprogramación a los sistemas computacionales, lo que trajo una ventaja importante en términos de eficiencia. ¿Por qué la introducción de la multiprogramación fue una ventaja?

   1.3) **[2p]** En los sistemas modernos, parte del procesamiento de un cambio de contexto se escribe en *assembler*, en lugar de un lenguaje de alto nivel (como C). ¿Por qué?

   1.4) **[2p]** ¿Es posible implementar un sistema operativo sin utilizar *timer*s? Justifique su respuesta.

   1.5) **[2p]** ¿Qué diferencia hay entre los sistemas *batch* y los sistemas interactivos?

   1.6) **[2p]** ¿Qué diferencia hay entre las señales `SIGTERM` y `SIGKILL`, que pueden ser enviadas con la *syscall* `kill`?

   1.7) **[6p]** Se desea introducir un nuevo tipo de interfaz: LightUSB, para lo cual el sistema operativo debe ser capaz de reconocer estos dispositivos y utilizar sus *drivers*. Suponiendo que posee el código fuente de los *drivers* necesarios, ¿qué pasos debería seguir para incorporar los *drivers* a su sistema operativo? Mencione una ventaja y una desventaja para cada uno de estos casos

   a) Sistema monolítico
   b) Sistema monolítico con módulos
   c) Sistema basado en microkernel

2. **[9p]** Se proponen dos alternativas para implementar una instrucción que bloquea un proceso por `T` segundos, donde `GetTime()` es una función que retorna un entero con el tiempo actual (un *timestamp*) en segundos.

```c
void blockFor(unsigned int T) {
    unsigned int t = GetTime();
    while (GetTime() - t < T)
        ; // do nothing in loop
}
```

```c
#include <unistd.h>
void blockFor(unsigned int T) {
    sleep(T);
}
```

Describa la diferencia entre ambas alternativas en términos de utilización de CPU, estado de los procesos, y comportamiento en las colas del sistema operativo.

3. **[15p]** De acuerdo al estado del sistema presentado en la hoja adjunta, responda las siguientes preguntas:

   3.1) **[3p]** En la situación A, los usuarios `cruz` y `jlopez` están ejecutando el proceso `tom`, el cual hace uso intensivo de CPU. ¿Cómo puede explicar el hecho que haya varios procesos en ejecución y todos posean 100 % de uso de CPU?

   3.2) **[4p]** En la situación B, el usuario `jheysen` ha iniciado también el proceso `tom`. ¿Por qué esto hace que el uso de CPU de todos los procesos baje? El proceso `tom` se ejecutó de la misma manera, y con los mismos argumentos que utilizaron los usuarios `cruz` y `jlopez`.

   3.3) **[4p]** Dibuje, con la información disponible, un árbol de procesos para la situación B. Para cada proceso incluya el par $\langle$ *PID*, *command* $\rangle$

   3.4) **[4p]** En la situación C, solo el usuario `cruz` se encuentra ejecutando sus procesos. Además, él se encuentra ejecutando dos procesos `jerry`. ¿Por qué estos procesos no consumen CPU?

4. **[15p]** Se solicita escribir un programa *duplicador de comandos* que cumpla los siguientes requisitos. R1: El programa debe leer dos comandos y sus respectivos argumentos. R2: el programa debe ejecutar ambos comandos **en paralelo** (esto es, que puedan estar en la cola *ready* al mismo tiempo), y luego esperar más comandos. R3: si algún comando termina de manera anormal (*exit code* distinto a 0), debe reejecutarlo solo una vez más, y luego imprimir los PID de aquellos que fallaron ambas veces. R4: el programa no debe terminar si uno de los procesos falla.

```c
// suponga que todos los headers necesarios han sido incluidos
char *command1; char *command2;
char *arguments1; char *arguments2
int *exitCode; pid_t p1, p2;
// puede agregar mas variables aqui
while('r') {
  // lee correctamente ambos comandos y argumentos
  read_command(&command1, &arguments1, &command2, &arguments2);
  exec(command1, arguments1);
  if ( p1 = fork() > 0 ) { waitpid(p, &exitCode); }
  exec(command2, arguments2);
  if ( p2 = fork() > 0 ) { waitpid(p, &exitCode); }
}
```

   4.1) **[4p]** El fragmento de código presentado no cumple con todos los requisitos. Explique cuáles **NO** cumple y por qué.

   4.2) **[7p]** Modifique el código presentado para que cumpla todos los requisitos.

   4.3) **[4p]** Si eliminamos el requisito R3, ¿cómo podría modificar el código del ítem anterior para que los procesos creados quedaran en estado *zombie*? No es necesario que escriba el código, sino indicar las modificaciones (si sale más sencillo reescribirlo, reescríbalo). ¿Qué implicancias tendría esto para el sistema operativo?

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`.

Situación A

```
top - 09:18:43 up 27 days, 17:12,  5 users,  load average: 0.77, 0.60, 0.64
Tasks: 164 total,   5 running, 159 sleeping,   0 stopped,   0 zombie
%Cpu(s):100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st

  PID USER      PR    VIRT    RES    SHR S  %CPU %MEM     TIME+  PPID COMMAND
25927 cruz      20    6520    724    644 R 100.0  0.0   0:10.69 25587 tom
25928 cruz      20    6520    756    676 R 100.0  0.0   0:10.69 25587 tom
25930 jlopez    20    6520    680    600 R 100.0  0.0   0:07.47 25879 tom
25929 jlopez    20    6520    720    636 R  99.7  0.0   0:07.45 25879 tom
    1 root      20   37644   5736   4008 S   0.0  0.1   0:56.78     0 systemd
    2 root      20       0      0      0 S   0.0  0.0   0:00.50     0 kthreadd
25586 cruz      20   95480   3352   2316 S   0.0  0.0   0:00.10 25504 sshd
25587 cruz      20   22580   5444   3560 S   0.0  0.1   0:00.20 25586 bash
25862 cruz      20   22572   5192   3324 S   0.0  0.1   0:00.11 25861 bash
25877 root      20   52704   3928   3464 S   0.0  0.0   0:00.00 25862 sudo
25878 root      20   52284   3552   3136 S   0.0  0.0   0:00.00 25877 su
25879 jlopez    20   22616   5424   3504 S   0.0  0.1   0:00.12 25878 bash
25911 cruz      20    4356    688    612 S   0.0  0.0   0:00.00 25687 jerry
25912 cruz      20    4356     84      0 S   0.0  0.0   0:00.00 25911 jerry
```

Situación B

```
top - 09:21:51 up 27 days, 17:16,  5 users,  load average: 4.84, 2.45, 1.36
Tasks: 166 total,   7 running, 159 sleeping,   0 stopped,   0 zombie
%Cpu(s):100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st

  PID USER      PR    VIRT    RES    SHR S  %CPU %MEM     TIME+  PPID COMMAND
25933 jheysen   20    6520    756    672 R  72.1  0.0   0:25.98 25815 tom
25927 cruz      20    6520    724    644 R  65.4  0.0   3:05.27 25587 tom
25928 cruz      20    6520    756    676 R  65.4  0.0   3:06.59 25587 tom
25932 jheysen   20    6520    720    636 R  61.1  0.0   0:23.44 25815 tom
25929 jlopez    20    6520    720    636 R  58.5  0.0   3:00.88 25879 tom
    1 root      20   37644   5736   4008 S   0.0  0.1   0:56.78     0 systemd
    2 root      20       0      0      0 S   0.0  0.0   0:00.50     0 kthreadd
    3 root      20       0      0      0 S   0.0  0.0   0:00.70     2 ksoftirqd/0
25800 cruz      20   95404   3236   2300 S   0.0  0.0   0:00.03 25769 sshd
25801 cruz      20   22572   5256   3384 S   0.0  0.1   0:00.10 25800 bash
25813 root      20   52932   3956   3380 S   0.0  0.0   0:00.02 25801 sudo
25814 root      20   52284   3508   3088 S   0.0  0.0   0:00.00 25813 su
25815 jheysen   20   22672   5260   3316 S   0.0  0.1   0:00.12 25814 bash
25830 root      20   95480   7132   6096 S   0.0  0.1   0:00.04  1024 sshd
25861 cruz      20   95480   3436   2400 S   0.0  0.0   0:00.04 25830 sshd
25862 cruz      20   22572   5192   3324 S   0.0  0.1   0:00.11 25861 bash
25877 root      20   52704   3928   3464 S   0.0  0.0   0:00.00 25862 sudo
25878 root      20   52284   3552   3136 S   0.0  0.0   0:00.00 25877 su
25879 jlopez    20   22616   5424   3504 S   0.0  0.1   0:00.12 25878 bash
25911 cruz      20    4356    688    612 S   0.0  0.0   0:00.00 25687 jerry
25912 cruz      20    4356     84      0 S   0.0  0.0   0:00.00 25911 jerry
25917 cruz      20   41800   3736   3104 R   0.0  0.0   0:02.24 25643 top
```

Situación C

```
top - 09:28:59 up 27 days, 17:23,  5 users,  load average: 4.99, 4.92, 3.00
Tasks: 162 total,   3 running, 159 sleeping,   0 stopped,   0 zombie
%Cpu(s): 50.0 us,  0.0 sy,  0.0 ni, 50.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st

  PID USER      PR    VIRT    RES    SHR S  %CPU %MEM     TIME+  PPID COMMAND
25927 cruz      20    6520    724    644 R 100.0  0.0   7:56.72 25587 tom
25928 cruz      20    6520    756    676 R 100.0  0.0   7:56.85 25587 tom
    1 root      20   37644   5736   4008 S   0.0  0.1   0:56.78     0 systemd
    2 root      20       0      0      0 S   0.0  0.0   0:00.50     0 kthreadd
25504 root      20   95480   6896   5860 S   0.0  0.1   0:00.05  1024 sshd
25506 cruz      20   45248   4744   4044 S   0.0  0.1   0:00.03     1 systemd
25586 cruz      20   95480   3352   2316 S   0.0  0.0   0:00.16 25504 sshd
25587 cruz      20   22580   5444   3560 S   0.0  0.1   0:00.20 25586 bash
25611 root      20   95404   6868   5924 S   0.0  0.1   0:00.03  1024 sshd
25642 cruz      20   95404   3256   2316 S   0.0  0.0   0:00.77 25611 sshd
25643 cruz      20   22572   5188   3316 S   0.0  0.1   0:00.10 25642 bash
25655 root      20   95404   6808   5868 S   0.0  0.1   0:00.02  1024 sshd
25686 cruz      20   95404   3116   2180 S   0.0  0.0   0:00.00 25655 sshd
25687 cruz      20   22572   5124   3252 S   0.0  0.1   0:00.10 25686 bash
25911 cruz      20    4356    688    612 S   0.0  0.0   0:00.00 25687 jerry
25912 cruz      20    4356     84      0 S   0.0  0.0   0:00.00 25911 jerry
25917 cruz      20   41800   3736   3104 R   0.0  0.0   0:03.65 25643 top
```