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

IIC2333 — Sistemas Operativos y Redes — 1/2025

Tarea 0

Miércoles 19-Marzo-2025

Fecha de Entrega: Miércoles 02-Abril-2025 a las 21:00
Composición: Tarea en Parejas

Objetivos

- Utilizar *syscalls* para construir un programa que administre el ciclo de vida de un conjunto de procesos.
- Establecer un mecanismo de comunicación entre procesos mediante señales.

`DCCAdmin`

Se deberá desarrollar un programa denominado **DCCAdmin**, diseñado para ejecutar múltiples programas de manera concurrente mediante la creación de procesos. Además, deberá supervisar y garantizar que ningún proceso exceda un tiempo de ejecución determinado. Para lograrlo, se implementará un intérprete de comandos, también conocido como *shell*, que deberá cumplir con los siguientes requisitos:

- El programa principal **DCCAdmin** no debe bloquearse. Se espera que los programas externos se ejecuten **en paralelo** mediante la creación de múltiples procesos.
- Permitir el envío de señales a distintos procesos según sea necesario.
- Es obligatorio el uso de las *syscalls* vistas en clases.
- Garantizar que no queden procesos *zombie* o huérfanos en ningún caso de uso.

Funcionalidad del programa

Las principales funciones que debe cumplir el programa son:

1. El programa deberá manejar diversos comandos y ejecutar múltiples programas externos **de manera paralela** mediante la creación de procesos independientes. Además, deberá proporcionar el *feedback* correspondiente a cada comando cuando sea necesario.
2. Cuando un proceso creado alcance el tiempo límite $\langle time\_max \rangle$, se le enviará una señal `SIGTERM` para notificarle que tiene 5 segundos para finalizar su ejecución. En caso de no terminar después de este tiempo, se enviará la señal `SIGKILL` para forzar su terminación. Esta acción solo ocurrirá si hay procesos en ejecución. Es importante destacar que el programa principal continuará procesando comandos mientras esto sucede.
3. Al finalizar el programa principal, se tiene que mostrar en consola las estadísticas de los procesos ejecutados.

Comandos del programa

Los comandos que su *shell* debe soportar son:

- `start <executable> <arg1> <arg2> ... <argn>`: Este comando toma la ruta *executable* de un ejecutable y los argumentos *argi* correspondientes a este ejecutable y lo ejecuta mediante un nuevo proceso, distinto al de la *shell*. En caso de que el ejecutable no exista, se le debe indicar el error al usuario. **Correr uno o varios programas no debe congelar su *shell***. Deberá investigar qué *syscalls* utilizar para cumplir con este requerimiento.
- `info`: Este comando se encarga de entregar al usuario un listado de todos los programas que fueron ejecutados desde `dccadmin` y se encuentran ejecutando en un determinado momento. A continuación, se detalla la información mínima a mostrar.
  - PID del proceso
  - Nombre del ejecutable
  - Tiempo de ejecución del proceso en segundos [1]
  - `exit code` en caso de que el proceso hijo haya terminado, -1 en caso contrario.
  - `signal value`: Valor de la señal recibida por el proceso. Si no recibió ninguna, será -1. Por ejemplo, si se emite una señal `SIGINT`, el estado sería 2.
- `timeout <time>`: Este comando permite terminar todos los procesos que se estén ejecutando de manera concurrente en el programa principal en ese momento. Antes de ejecutarse, se debe validar si existen procesos en ejecución.
  - Si no hay procesos en ejecución, se debe informar al usuario con el siguiente mensaje:

    ```
    No hay procesos en ejecución. Timeout no se puede ejecutar.
    ```

  - En caso de que sí haya procesos en ejecución, se esperará hasta que transcurra el tiempo definido en `<time>`. Si un proceso no finaliza dentro de ese tiempo, se debe imprimir la siguiente información antes de enviar la señal `SIGTERM`:

    ```
    Timeout cumplido!
    PID nombre_del_ejecutable tiempo_ejecución exit_code signal_value
    ```

- `quit`: Este comando permite terminar el programa principal `dccadmin`. En caso de que aún existan procesos ejecutándose, se tiene que enviar `SIGINT` a los procesos correspondientes. En el caso de que los procesos no terminen pasados 10 segundos, se eliminarán por medio de una señal `SIGKILL`. Por último, al finalizar el programa principal se imprimen las estadísticas de los procesos.

  ```
  DCCAdmin finalizado
  PID nombre_del_ejecutable tiempo_ejecución exit_code signal_value
  ```

  Tener en cuenta que si el usuario intenta terminar con el programa mediante el envío de una señal `SIGINT` (Ctrl + C), el programa debe comportarse **exactamente igual** a que si se hubiera usado este comando.

[1] Para obtener el tiempo en segundos pueden utilizar time

Supuestos

Puedo realizar los siguientes supuestos:

- Siempre se entregará un comando válido.
- Siempre se entregarán la cantidad de argumentos requeridos.
- No es necesario considerar el caso en que $\langle time\_max \rangle$ y $\langle timeout \rangle$ finalicen exactamente al mismo tiempo.
- Por cada ejecución de `dccadmin` se ejecutarán a lo más 10 programas con `start`.

Ejecución

El programa principal será ejecutado por línea de comandos con la siguiente sintaxis:

```
./dccadmin [<time_max>]
```

Donde:

- `<time_max>` es un parámetro **opcional**, que es un entero que indica la cantidad máxima de segundos que puede demorarse en correr un proceso antes de que sea terminado. Si no se entrega este parámetro, se debe considerar que los procesos tienen tiempo ilimitado para ejecutar.

Ahora que se ha ejecutado `dccadmin` tu programa debe esperar por comandos del usuario. Un ejemplo de los comandos que podrían ser ejecutados es:

```
start sleep 20
info
quit
```

Formalidades

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso[2]. Para entregar su tarea usted deberá crear una carpeta llamada `T0` en el directorio principal de su carpeta personal y subir su tarea a esa carpeta. En su carpeta `T0` **solo debe incluir el código fuente** necesario para compilar su tarea, además del reporte y un `Makefile`. Se revisará el contenido de dicha carpeta el día Miércoles 02-Abril-2025 a las 21:00.

- La tarea debe ser realizada solamante en parejas.
- La tarea deberá ser realizada en el lenguaje de programación **C**. Cualquier tarea escrita en otro lenguaje de programación no será revisada.
- **NO debe incluir archivos binarios**[3]. En caso contrario, tendrá un descuento de 0.2 décimas en su nota final.
- **NO debe incluir un repositorio de git**[4]. En caso contrario, tendrá un descuento de 0.2 décimas en su nota final.
- **NO debe usar VSCode para entrar al servidor**[5]. En caso contrario, tendrá un descuento de 0,2 puntos en su nota final.
- Si inscribe de forma incorrecta su grupo o no lo inscribe, tendrá un descuento de 0.3 décimas

---

2 `iic2333.ing.puc.cl`
3 Los archivos que resulten del proceso de compilar, como lo son los ejecutables y los *object files*
4 Si es que lo hace, puede eliminar la carpeta oculta `.git` antes de la fecha de entrega.
5 Si es que lo hace, puede eliminar la carpeta oculta `.vscode-server` antes de la fecha de entrega.

- Su tarea debe compilarse utilizando el comando `make`, y generar un ejecutable llamado `dccadmin` en esa misma carpeta. Si su programa **no tiene** un `Makefile`, tendrá un descuento de 0.5 décimas en su nota final y corre riesgo que su tarea no sea corregida.
- En caso de no cumplir con el formato de entrega, tendrá un descuento de 0.3 décimas en la nota final.
- Si ésta **no compila** o **no funciona** (*segmentation fault*), obtendrán la nota mínima, pudiendo recorregir modificando líneas de código con un descuento de una décima por cada cuatro líneas modificada, con un máximo de 20 lineas a modificar.

El no respeto de las formalidades o un código extremadamente desordenado podría originar descuentos adicionales, los cuales quedarán a discreción del ayudante corrector. Se recomienda modularizar, utilizar funciones y ocupar nombres de variables explicativos. En el caso de no entregar en la carpeta especificada la tarea, **no** se corregirá.

Evaluación

- **1.5 pts.** Correcta implementación de `start` con múltiples procesos paralelos.
- **0.7 pts.** Correcta implementación de *time_max*.
- **1.0 pts.** Correcta implementación de `info`.
- **1.3 pts.** Correcta implementación de `timeout`.
- **0.5 pts.** Correcta implementación de `quit`.
- **0.5 pts.** Correcta implementación de Ctrl + C .
- **0.5 pts.** Manejo de memoria. Se otorgará este puntaje si `valgrind` reporta 0 *leaks* y 0 errores de memoria en **todos los casos de uso**[6]. Para esta tarea, se permitirá como única fuga de memoria la variable que almacena el comando.

Política de atraso

Se puede hacer entrega de la tarea con un máximo de 2 días hábiles de atraso. La fórmula a seguir es la siguiente:

$$N_{T_1}^{\text{Atraso}} = \text{mín}(N_{T_1}, 7{,}0 - 0{,}75 \cdot d)$$

Siendo d la cantidad de días de atraso. Notar que esto equivale a un descuento *soft*, es decir, cambia la nota máxima alcanzable y no se realiza un descuento directo sobre la nota obtenida. El uso de días de atraso no implica días extras para alguna tarea futura, por lo que deben usarse bajo su propio riesgo.

Importante: Se recomienda incluir un archivo **README.md** en el que se indiquen las funcionalidades implementadas y/o cualquier consideración que deba tenerse en cuenta al momento de corregir la tarea. Además, en caso de no cumplir con el formato de entrega, tendrá un descuento de 0.3 décimas en la nota final.

Preguntas

Cualquier duda preguntar a través del foro oficial.

[6] Es decir, el código debe mostrar 0 *leaks* y 0 errores en todas las pruebas.