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

IIC2333 — Sistemas Operativos y Redes — 1/2022

Tarea 1

Viernes 18-Marzo-2022

Fecha de Entrega: Lunes 28-Marzo-2022 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.
- Comunicar múltiples procesos por medio del uso de señales.

`crsh`

Después de años trabajando con computadores, el gran **Cruz** está aburrido de usar todos los días la misma terminal para correr los trabajos de sus alumnos, sus múltiples redes neuronales, abrir ~~`vim`~~ `nano`, y eliminar ~~los memes~~ el *spam* que le envían sus ayudantes de Sistemas Operativos.

Es por esto que el profesor recurre a su ayuda para solucionar su problema y se le ocurre una gran idea: **`crsh`** (Cristian Ruz SHell), la cual usa **multiprogramación** para correr múltiples programas a la vez y poder subir su velocidad de corrección de tareas en un 420 %.

Requisitos

Deberá implementar un intérprete de comandos, también conocido como *shell*, que cumpla con los siguientes requisitos:

- Ejecutar programas externos por medio de la creación de procesos hijos.
- Monitorear el ciclo de vida de procesos hijos.
- Enviar señales a diferentes procesos.

Funcionalidad del programa

Su programa deberá ser capaz de recibir múltiples comandos y entregar el *feedback* correspondiente a cada uno de estos de ser necesario.

Los comandos que su *shell* deberá ser capaz de soportar son:

- `hello`: Este comando crea un nuevo proceso, distinto al de la *shell* que imprime en la consola el *string* `Hello World!`.
- `sum` `<num_1>` `<num_2>`: Este comando crea un nuevo proceso, distinto al de la *shell* que toma como *input* dos números de punto flotante e imprime en la consola la suma de estos.

- `is_prime <num>`: Este comando crea un nuevo proceso, distinto al de la *shell* que toma como *input* un número entero y calcula si es primo, imprimiendo en consola un mensaje que indique el resultado.[1]
- `crexec <executable> <input>`: Este comando toma el nombre de un ejecutable y el *input* correspondiente 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***.
- `crlist`: Este comando se encarga de entregar al usuario un listado de todos los programas que fueron ejecutados desde `crsh` y se encuentran ejecutando en un determinado momento. En esta lista deben ir datos como:
  - PID del proceso
  - Nombre del ejecutable
  - Tiempo de ejecución del proceso en segundos [2]
- `crexit`: Este comando termina la ejecución de su programa. Anterior a esto, deben enviar un `SIGINT` a cada proceso hijo y esperar que todos terminen. En el caso que los programas no terminen pasados 15 segundos, se eliminarán los procesos por medio de una señal `SIGKILL`.

  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.

Ejecución

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

```
./crsh
```

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

```
crexec helloworld

crexec average 7 8

crlist

sum 3 5
```

Estos ejemplos funcionan correctamente si suponemos que existen los ejecutables `helloworld` y `average`.

Formalidades

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso[3]. Para entregar su tarea usted deberá crear una carpeta llamada **exactamente** `T1`[4] en el directorio principal de su carpeta personal y subir su tarea a esa carpeta. En su carpeta `T1` **solo debe incluir el código fuente** necesario para compilar su tarea y un `Makefile`. Se revisará el contenido de dicha carpeta el día Lunes 28-Marzo-2022 a las 21:00.

- La tarea debe ser realizada solamante en parejas.

---

1 Tienen libertad de decidir el mensaje a imprimir.
2 Para obtener el tiempo en segundos pueden utilizar time
3 `iic2333.ing.puc.cl`
4 Se debe respetar el uso de mayúsculas, sino, la tarea no sera recolectada.

- 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**[5]. En caso contrario, tendrá un descuento de 0,2 puntos en su nota final.
- **NO debe incluir un repositorio de git**[6]. En caso contrario, tendrá un descuento de 0,2 puntos en su nota final.
- **NO debe usar VSCode para entrar al servidor**[7]. En caso contrario, tendrá un descuento de 0,2 puntos en su nota final.
- Su tarea **debe encontrarse** en la carpeta `T1`, compilarse utilizando el comando `make`, y generar un ejecutable llamado `crsh` en esa misma carpeta. Si su programa **no tiene** un `Makefile`, tendrá un descuento de 0,5 décimas en su nota final.
- Es muy importante que su tarea corra dentro del servidor del curso. 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

- **0.20 pts.** Correcta implementación de `hello`.
- **0.20 pts.** Correcta implementación de `sum`.
- **0.60 pts.** Correcta implementación de `is_prime`.
- **1.50 pts.** Correcta implementación de `crexec`.
- **1.25 pts.** Correcta implementación de `crlist`.
- **1.25 pts.** Correcta implementación de `crexit`.
- **1.00 pts.** Manejo de memoria. Se obtiene este puntaje si `valgrind` reporta en su código 0 *leaks* y 0 errores de memoria en **todo caso de uso**[8]

Política de atraso

Se puede hacer entrega de la tarea con un máximo de 4 días 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.

Para poder hacer entrega atrasada se debe responder el formulario de atraso.

Preguntas

Cualquier duda preguntar a través del foro oficial.

---

5 Los archivos que resulten del proceso de compilar, como lo son los ejecutables y los *object files*
6 Si es que lo hace, puede eliminar la carpeta oculta `.git` antes de la fecha de entrega.
7 Si es que lo hace, puede eliminar la carpeta oculta `.vscode-server` antes de la fecha de entrega.
8 Es decir, debe reportar 0 *leaks* y 0 errores para todo *test*.