Arquitectura de Sistemas

Práctica 4: Procesos

Gustavo Romero López

Updated: 14 de febrero de 2019

Arquitectura y Tecnología de Computadores

Objetivos

- Aprender a gestionar procesos:
  - creación: `fork()`.
  - cambio de la imagen: `exec()`.
  - terminación: `exit()`.
  - comunicación: `exit()` + `wait()`.
  - identificador de proceso: `getpid()`.
- Medición de tiempos.
- Comunicación entre procesos padre/hijo.
- Comunicación mediante memoria compartida (opcional).
- Comunicación mediante paso de mensajes (opcional).

https://pccito.ugr.es/~gustavo/as/practicas/04/makefile

```
SRC = $(wildcard *.c *.cc)
EXE = $(basename $(SRC))

CFLAGS = -march=native -O3 -pthread -Wall
CXXFLAGS = $(CFLAGS) -std=c++11
LDFLAGS = -lpthread -lrt

default: $(EXE)

clean:
  -rm -fv $(EXE) *~ core.*

.PHONY: clean default
```

Creación de un proceso nuevo

- Mediante `fork()` podemos crear un nuevo proceso.
- El nuevo proceso es una copia idéntica del proceso original, indistinguible salvo por el identificador de proceso. Este se puede consultar mediante `getpid()`.
- Esta llamada la sistema tiene la peculiaridad de que es ejecutada desde un único proceso pero retorna a dos: el original, denominado **proceso padre** y, si todo ha ido bien, otro nuevo proceso denominado **proceso hijo**.
- El valor de retorno de `fork()` es:
  - **-1** en caso de error.
  - **0** en el proceso hijo.
  - **El identificador del proceso hijo** en el proceso padre.
- Pruebe el siguiente programa de ejemplo:

https://pccito.ugr.es/~gustavo/as/practicas/04/fork.cc

```
//------------------------------------------------------
// fork.cc
//------------------------------------------------------

#include <unistd.h>
#include <iostream>

//------------------------------------------------------

int main()
{
  switch(fork())
  {
    case -1: std::cout << "fallo en fork()!"; break;
    case  0: std::cout << "hijo";             break;
    default: std::cout << "padre";            break;
  }
  std::cout << "\t [" << getpid() << "]" << std::endl;
}

//------------------------------------------------------
```

Medición de tiempos (1)

Métodos para medir tiempo:

- `std::clock::high_resolution_clock` $\Longrightarrow$ precisión: nanosegundos.
- `clock_gettime` $\Longrightarrow$ precisión: nanosegundos.
- `gettimeofday` $\Longrightarrow$ precisión: microsegundos.
- `getrusage` $\Longrightarrow$ precisión: microsegundos.
- `clock` $\Longrightarrow$ precisión: ticks del reloj.
- `time` $\Longrightarrow$ precisión: milisegundos.

Medición de tiempos (2)

- Modifique el programa anterior de forma que pueda medir el tiempo que se tarda en ejecutar un proceso nulo.
- El proceso nulo es tan sencillo como: `int main() {}`.
- Además de `fork()`, ahora deberá utilizar las llamadas a `wait()` y `exec()`.
- Ahora el proceso padre deberá esperar a que acabe el proceso hijo con `wait()`.
- El proceso hijo cambiará su imagen para ejecutar el proceso nulo mediante `execl()`.

Comunicación entre procesos padre/hijo

- La comunicación más básica entre los procesos padre/hijo es que el valor de retorno del proceso hijo es comunicado al proceso padre.
  - El proceso hijo devuelve el valor mediante `exit()` que coincide con el valor devuelto por `main()` cuando no se utiliza explícitamente `exit()`.
  - El proceso padre puede recuperar el valor devuelto por el hijo mediante `wait()` y `waitpid()`. Esta segunda versión permite escoger el hijo con el que comunicarse en caso de que existan varios.
- Modifique el programa de ejemplo original fork.cc para que el proceso padre reciba un valor desde el proceso hijo.
- El valor recibido lleva más información que el valor que se desea comunicar. Para descartar el resto podemos utilizar la función `WEXITSTATUS()`.

- Busque información sobre las funciones `shmget()` y `shmctl()` de la cabecera `#include <sys/shm.h>`.
- Escriba un programa que envíe un mensaje entre dos procesos haciendo uso de memoria compartida.

- El proceso es mucho más sencillo combinando el uso de `fork()` y `mmap()` para conseguir crear los dos procesos y disponer de un area de memoria compartida.

Comunicación entre procesos mediante paso de mensajes

- Busque información sobre las funciones `msgctl()`, `msgget()`, `msgsnd()` y `msgrcv()` de la cabecera `#include <sys/msg.h>`.
- Escriba un programa que envíe un mensaje entre dos procesos haciendo uso de este tipo de paso de mensajes.

ping/pong

- Para estudiar y practicar la comunicación entre procesos vamos a implementar el protocolo ping/pong:
  - Dos programas deben enviar y recibir mensajes...
  - Servidor: espera recibir ping y responde con pong.
  - Cliente: envía ping y espera recibir pong.
- Mecanismos de implementación (sin ejemplo):
  - Memoria compartida entre procesos: `mmap`.
  - Pasos de mensajes (System V): `msgget/msgsnd`.
  - Colas de mensajes (POSIX): `mq_open/mq_send/mq_receive`.
  - Tuberías: `pipe`.
  - Memoria compartida entre hebras: `pthreads/C++11`.
  - Sockets: `socket/bind/listen/read/write/close`.
- Estudie los códigos de muestra y cree su propio programa siguiendo el mismo esquema: `https://pccito.ugr.es/~gustavo/as/practicas/ping-pong`.

Comunicación entre procesos mediante sockets