UNIVERSIDAD NACIONAL DE LA PLATA

Sistemas Operativos

Práctica 3 (2025)

Requisitos

Para realizar esta práctica se puede usar la misma máquina virtual de la práctica 1 o una de su elección si resulta más cómodo (por ejemplo una VM con interfaz gráfica y un IDE).

Threading (ULT y KLT)

Conceptos generales

1. ¿Cuál es la diferencia fundamental entre un proceso y un thread?
2. ¿Qué son los User-Level Threads (ULT) y cómo se diferencian de los Kernel-Level Threads (KLT)?
3. ¿Quién es responsable de la planificación de los ULT? ¿y los KLT? ¿Cómo afecta esto al rendimiento en sistemas con múltiples núcleos?
4. ¿Cómo maneja el sistema operativo los KLT y en qué se diferencian de los procesos?
5. ¿Qué ventajas tienen los KLT sobre los ULT? ¿Cuáles son sus desventajas?
6. Qué retornan las siguientes funciones:
    a. getpid()
    b. getppid()
    c. gettid()
    d. pthread_self()
    e. pth_self()
7. ¿Qué mecanismos de sincronización se pueden usar? ¿Es necesario usar mecanismos de sincronización si se usan ULT?
8. Procesos
    a. ¿Qué utilidad tiene ejecutar fork() sin ejecutar exec()?
    b. ¿Qué utilidad tiene ejecutar fork() + exec()?
    c. ¿Cuál de las 2 asigna un nuevo PID fork() o exec()?
    d. ¿Qué implica el uso de Copy-On-Write (COW) cuando se hace fork()?
    e. ¿Qué consecuencias tiene no hacer wait() sobre un proceso hijo?
    f. ¿Quién tendrá la responsabilidad de hacer el wait() si el proceso padre termina sin hacer wait()?
9. Kernel Level Threads
    a. ¿Qué elementos del espacio de direcciones comparten los threads creados con pthread_create()?
    b. ¿Qué relaciones hay entre getpid() y gettid() en los KLT?
    c. ¿Por qué pthread_join() es importante en programas que usan múltiples hilos? ¿Cuándo se liberan los recursos de un hilo zombie?
    d. ¿Qué pasaría si un hilo del proceso bloquea en read()? ¿Afecta a los demás hilos?
    e. Describí qué ocurre a nivel de sistema operativo cuando se invoca pthread_create() (¿es syscall? ¿usa clone?).
10. User Level Threads

a. ¿Por qué los ULTs no se pueden ejecutar en paralelo sobre múltiples núcleos?
b. ¿Qué ventajas tiene el uso de ULTs respecto de los KLTs?
c. ¿Qué relaciones hay entre getpid(), gettid() y pth_self() (en GNU Pth)?
d. ¿Qué pasaría si un ULT realiza una syscall bloqueante como read()?
e. ¿Qué tipos de scheduling pueden tener los ULTs? ¿Cuál es el más común?

11. Global Interpreter Lock
    a. ¿Qué es el GIL (Global Interpreter Lock)? ¿Qué impacto tiene sobre programas multi-thread en Python y Ruby?
    b. ¿Por qué en CPython o MRI se recomienda usar procesos en vez de hilos para tareas intensivas en CPU?

Práctica guiada

1. Instale las dependencias necesarias para la práctica (strace, git, gcc, make, libc6-dev, libpth-dev, python3, htop y podman):

```
apt update
apt install build-essential libpth-dev python3 python3-venv strace git
htop podman
```

2. Clone el repositorio con el código a usar en la práctica

```
git clone https://gitlab.com/unlp-so/codigo-para-practicas.git
```

3. Resuelva y responda utilizando el contenido del directorio practica3/01-strace:
    a. Compile los 3 programas C usando el comando make.
    b. Ejecute cada programa individualmente, observe las diferencias y similitudes del PID y THREAD_ID en cada caso. Conteste en qué mecanismo de concurrencia las distintas tareas:
        i. Comparten el mismo PID y THREAD_ID
        ii. Comparten el mismo PID pero con diferente THREAD_ID
        iii. Tienen distinto PID
    c. Ejecute cada programa usando strace (strace ./nombre_programa > /dev/null) y responda:
        i. ¿En qué casos se invoca a la systemcall clone o clone3 y en cuál no? ¿Por qué?
        ii. Observe los flags que se pasan al invocar a clone o clone3 y verifique en qué caso se usan los flags CLONE_THREAD y CLONE_VM.
        iii. Investigue qué significan los flags CLONE_THREAD y CLONE_VM usando la manpage de clone y explique cómo se relacionan con las diferencias entre procesos e hilos.
        iv. printf() eventualmente invoca la syscall write (con primer argumento 1, indicando que el file descriptor donde se escribirá el texto es STDOUT). Vea la salida de strace y verifique qué invocaciones a write(1, ...) ocurren en cada caso.
        v. Pruebe invocar de nuevo strace con la opción -f y vea qué sucede respecto a las invocaciones a write(1, …). Investigue qué es esa opción en la manpage de strace. ¿Por qué en el caso del ULT se puede ver la invocación a write(1, …) por parte del thread hijo aún sin usar -f?

4. Resuelva y responda utilizando el contenido del directorio practica3/02-memory:
    a. Compile los 3 programas C usando el comando make.
    b. Ejecute los 3 programas.
    c. Observe qué pasa con la modificación a la variable number en cada caso. ¿Por qué suceden cosas distintas en cada caso?
5. El directorio practica3/03-cpu-bound contiene programas en C y en Python que ejecutan una tarea CPU-Bound (calcular el enésimo número primo).
    a. Ejecute htop en una terminal separada para monitorear el uso de CPU en los siguientes incisos.
    b. Ejecute los distintos ejemplos con make (usar make help para ver cómo) y observe cómo aparecen los resultados, cuánto tarda cada thread y cuanto tarda el programa completo en finalizar.
    c. ¿Cuántos threads se crean en cada caso?
    d. ¿Cómo se comparan los tiempos de ejecución de los programas escritos en C (ult y klt)?
    e. ¿Cómo se comparan los tiempos de ejecución de los programas escritos en Python (ult.py y klt.py)?
    f. Modifique la cantidad de threads en los scripts Python con la variable NUM_THREADS para que en ambos casos se creen solamente 2 threads, vuelva a ejecutar y comparar los tiempos. ¿Nota algún cambio? ¿A qué se debe?
    g. ¿Qué conclusión puede sacar respecto a los ULT en tareas CPU-Bound?
6. El directorio practica3/04-io-bound contiene programas en C y en Python que ejecutan una tarea que simula ser IO-Bound (tiene una llamada a sleep lo que permite interleaving de forma similar al uso de IO).
    a. Ejecute htop en una terminal separada para monitorear el uso de CPU en los siguientes incisos.
    b. Ejecute los distintos ejemplos con make (usar make help para ver cómo) y observe cómo aparecen los resultados, cuánto tarda cada thread y cuanto tarda el programa completo en finalizar.
    c. ¿Cómo se comparan los tiempos de ejecución de los programas escritos en C (ult y klt)?
    d. ¿Cómo se comparan los tiempos de ejecución de los programas escritos en Python (ult.py y klt.py)?
    e. ¿Qué conclusión puede sacar respecto a los ULT en tareas IO-Bound?
7. Diríjase nuevamente en la terminal a practica3/03-cpu-bound y modifique klt.py de forma que vuelva a crear 5 threads.
    a. Ejecute htop en una terminal separada para monitorear el uso de CPU en los siguientes incisos.
    b. Ejecute una versión de Python que tenga el GIL deshabilitado usando: `make run_klt_py_nogil` (esta operación tarda la primera vez ya que necesita descargar un container con una versión de Python compilada explícitamente con el GIL deshabilitado).
    c. ¿Cómo se comparan los tiempos de ejecución de klt.py usando la versión normal de Python en contraste con la versión sin GIL?
    d. ¿Qué conclusión puede sacar respecto a los KLT con el GIL de Python en tareas CPU-Bound?