

# Sistemas Operativos

Primer cuatrimestre 2021

Trabajo Práctico Nro. 1 (obligatorio): Inter Process Communication

## Introducción

El trabajo práctico consiste en aprender a utilizar los distintos tipos de IPCs presentes en un sistema POSIX. Para ello se implementará un sistema que distribuirá tareas de SAT solving entre varios procesos.

## Grupos

Se realizará con los grupos ya establecidos.

## Requerimientos

Desarrollar un sistema en C que resuelva múltiples fórmulas proposicionales en CNF de forma distribuida.

Para ello cuenta con procesos aplicación, vista, y esclavos

## Proceso Aplicación

- **DEBE** recibir por línea de comando los archivos a procesar, por ejemplo:  
`$ solve files/*`
- **DEBE** iniciar los procesos esclavos.
- **DEBE** distribuir una cierta cantidad (significativamente menor que el total) de archivos entre los esclavos, por ejemplo, si se requiere procesar 100 archivos entre 5 esclavos se pueden distribuir 2 archivos por esclavo inicialmente, es decir 10 archivos.
- Cuando un esclavo se libera, la aplicación le **DEBE** enviar **UN** nuevo archivo para procesar.
- **DEBE** recibir el resultado del procesamiento de cada archivo y **DEBE** agregarlo a un buffer **POR ORDEN DE LLEGADA**.
- Cuando inicia, **DEBE** esperar 2 segundos a que aparezca un proceso vista, si lo hace le comparte el buffer de llegada.
- **SOLO** debe imprimir por stdout el input que el proceso vista requiera para conectarse al buffer compartido (ver más adelante).
- Termina cuando todos los archivos estén procesados.
- **DEBE** guardar el resultado en el archivo resultado, aparezca el proceso de vista o no.

## Proceso Vista

- **DEBE** recibir por entrada estándar y como parámetro la información necesaria para conectarse al buffer compartido. Esto **DEBERÁ** permitir utilizar un pipe para iniciar el proceso aplicación y el vista: `./solve files/* | ./vista`, y también iniciar la aplicación y más tarde la vista: `./solve files/*` en una terminal o en background y `./vista <info>` en otra terminal o en foreground.
- **DEBE** mostrar en pantalla el contenido del buffer de llegada a medida que se va cargando el mismo. El buffer **DEBE** tener la siguiente información:
  - Nombre de archivo.
  - Cantidad de cláusulas.
  - Cantidad de variables.
  - Resultado (SAT | UNSAT).
  - Tiempo de procesamiento.
  - ID del esclavo que lo procesó.

## Proceso Esclavo

- **DEBE** recibir el/los paths de los archivos a procesar y **DEBE** iniciar el programa correspondiente para procesarlos (minisat).
- **DEBE** enviar la información relevante del procesamiento (ver proceso vista) al proceso aplicación.
- **NO DEBE** volcar el resultado de minisat a un archivo en disco, para luego leerlo desde el esclavo, **DEBERÁ** recibir el output de minisat utilizando algún mecanismo de IPC más sofisticado.
- No es necesario escribir un parser del output de minisat, se pueden utilizar comandos como grep, sed, awk, etc.
  - Hint: popen
  - Hint: `grep -o -e "Number of.*[0-9]\+" -e "CPU time.*" -e ".*SATISFIABLE"`

## IPCs y primitivas de sincronización requeridos

- Para conectar el proceso aplicación con sus esclavos **DEBERÁ** utilizar pipes o **named pipes**.
- Para compartir el buffer del proceso aplicación **DEBERÁ** usar **Shared Memory con Semáforos**.

## Consideraciones generales

- El sistema **DEBERÁ** estar libre de deadlocks, race conditions y busy waiting.
- El código **DEBERÁ** tener un Makefile para las tarea de compilación, aunque será sencillo deberá aprovechar el potencial de Makefile.

- Para el desarrollo **DEBERÁN** utilizar algún servicio de control de versiones desde el principio del desarrollo. No se admitirán repositorios sin un historial desde el comienzo del TP.
- El repositorio utilizado **NO DEBERÁ** tener binarios **NI** archivos de prueba.
- La compilación con todos los warnings activados (-Wall) **NO DEBERÁ** arrojar warnings.
- El análisis con Valgrind **NO DEBERÁ** reportar problemas de memoria en ninguno de los procesos implicados desarrollados por ustedes.
- El análisis con PVS-studio y cppcheck **NO DEBERÁ** reportar errores.

## Informe

Se deberá presentar un informe **EN FORMATO PDF**, en el cual se desarrollen **DE FORMA BREVE**, los siguientes puntos:

- Decisiones tomadas durante el desarrollo, por ejemplo, qué IPCs utilizaron, qué mecanismos de sincronización y por qué, etc.
- Un diagrama ilustrando cómo se relacionan los diferentes procesos.
- Instrucciones de compilación y ejecución.
- Limitaciones.
- Problemas encontrados durante el desarrollo y cómo se solucionaron.
- Citas de fragmentos de código reutilizados de otras fuentes.

## Entorno de desarrollo y ejecución

Es un requisito obligatorio tanto para el desarrollo y compilación como para la ejecución del sistema, utilizar la imagen provista por la cátedra:

```
docker pull agodio/itba-so:1.0
```

## Evaluación

La evaluación incluye y no se limita a los siguientes puntos:

- [1] Deadline.
- [4] Funcionalidad (Mandatorio).
- [3] Calidad de código.
- [1] Limpieza de recursos del sistema.
- [1] Informe.
- Defensa.

## Entrega

**Fecha:** 05/04/2020 hasta las 23:59.

Se habilitará la actividad "TP1" en el campus donde se podrá subir el material requerido. En caso de tener algún problema con la entrega en Campus, enviarlo por mail a los docentes a cargo de la clase práctica.

**Entregables:** Link del repositorio **ESPECIFICANDO** el hash del commit **Y** la rama correspondiente a la entrega y el informe.

**Defensa del trabajo práctico:** Individual y obligatoria el 05/04/2020.