**TP 1: Lo’ coronaví’**

**Programación de Redes**

**Integrantes: Fucci Marco, Corales Manuel**

**VARIANTE A**

En el siguiente informe pasará a desarrollarse el funcionamiento del programa realizado acerca de la simulación de contagios del COVID-19, y se dará respuesta a las preguntas que fueron solicitadas en el enunciado.

**Explicación del código**

**Función “main”**

En esta función, la cual toma el papel de principal y base para la simulación, se comenzaron declarando todas las variables que serían utilizadas luego para ejecutar de forma correcta el programa. Algunas tomaban valores generados y otras requerían que la información sea ingresada a través de la consola.

Dentro del bucle **for** principal se realiza la simulación del contagio, en este se verifica proceso es, si el principal o uno de los derivados, y luego se muestra en pantalla lo que corresponda según esto (si es el principal el pid y la cantidad de personas que infectó, si es otro a estos datos se le agrega por quién ha sido infectado).

Una vez fuera del for, se pasa a averiguar cual fue el día o los días, según sea el caso, en el que más contagiados hubo a través de condicionales **if.** En estos se van comparando los contagiados totales de cada día (luego son mostrados en pantalla) y, al mismo tiempo, en otra variable (**nsick**) se van sumando los contagiados acumulados día a día, para devolver este valor al final de la función con la cantidad total de contagiados que hubo al finalizar la simulación.

**Función “share\_mem”**

Esta función vino predefinida en el esqueleto del programa y se utiliza para reservar los espacios en memoria que se necesitarán compartir durante la ejecución el programa.

**Función “simulate”**

Dentro de esta función se simula, tal como indica su nombre, cuantas personas resultan contagiadas después de una reunión. Se define si una persona resulta contagiada o no luego de haberse juntado con un spreader a partir de un número generado aleatoriamente entre 0 y 1. También, cumpliendo con la variante indicada, hay posibilidades de que se genere un Super Spreader que cuando le toque a él dar la reunión infectará a todos los asistentes. Finalmente, todos los valores generados en esta parte son devueltos y utilizados en la función **main**.

**Preguntas “disparadoras”**

1. **¿Podría correrse la simulación en un sistema de procesador único? ¿Hace falta asumir alguna característica en tal caso?¿Qué cambiaría si se corre en un sistema multiprocesador?**

Si, la ejecución de esta simulación en un sistema de procesador único sería posible. Lo único que debería tenerse en cuenta en este caso es que los procesos se ejecutarían utilizando **concurrencia**, en el cual estos se irían turnando en el único núcleo disponible. En cambio, si se utilizara un sistema multiprocesador para la ejecución de este programa, sería posible aplicar el **paralelismo**, en el cual más de un proceso podría ejecutarse de forma simultánea al existir la posibilidad de que se distribuyan en los diversos núcleos disponibles.

1. **Si el sistema donde corro mi simulación utiliza un scheduler First-Come, First-Served, ¿podría pasar que se den ejecuciones donde nunca termine? Asumiendo que mi simulación convive con otros procesos en el sistema, y que se adopta un scheduler con política Shortest Job First con preemption, ¿podrían haber starvation?**

Estas situaciones podrían llegar a darse, pero solo si ocurre algún error durante la ejecución de la simulación, ya que el normal funcionamiento no lo permitiría. Una ejecución donde nunca termine podría llegar a ocurrir si el padre se queda esperando a un hijo que nunca da una señal de que ha terminado, por lo que no se podría continuar. En el segundo caso, al darse la posibilidad de que se conviva con otros procesos, podría pasar que un proceso externo con mayor prioridad y más carga ingrese a **running** antes que nuestra simulación y demore a esta durante un tiempo indefinido. Pero dado que el programa empieza y termina en un lapso determinado, no es posible que bloquee el procesador indefinidamente

1. **Dada la máquina de estados de un proceso, ¿cuál sería un "recorrido de estados" típico de un proceso de nuestra simulación?**

El recorrido de estados que tendría un proceso en nuestra simulación sería el siguiente. Primero se crea por un padre, por lo que estaría en **new;** luego pasa a la cola de listos, en el que su estado sería **ready;** una vez que el scheduler lo acepta y le permite ejecutarse, pasaría al estado **running;** por último, si todavía restan otros procesos para esperar quedaría en estado de **waiting**, luego pasa a **running** ejecutando lo que queda de código y por último el proceso sería retirado en estado **terminated**.

1. **En su implementación, ¿podría ocurrir que la simulación no se de en orden? Es decir, que algún proceso creado en el día t entregue su output luego de que otros procesos creados en un día k con k > t hagan el suyo.**

Sí, es posible, no se hizo ningún tipo de coordinación entre los procesos para presentar su output: Un proceso puede arrancar el loop de infectar a otras personas y uno de esos infectados ya puede comenzar a realizar su propia simulación y si es lo suficientemente “rápido” como para terminar antes que el padre, el proceso que lo infectó, podría dar su output primero, aunque a primera vista parece bastante improbable

1. **¿Utilizaron algún esquema de comunicación entre procesos? Si es así, describir su uso. ¿Podría pasar que dos procesos intenten modificar el mismo espacio de memoria concurrentemente? ¿Sería esto un problema?**

Si, se utilizaron funciones que permitían la comunicación entre procesos. Esta se llevó a cabo a través de una función creada, la cual tenía el nombre de **share\_mem,** cuyo objetivo era compartir memoria entre procesos mientras estos se encontraban en ejecución. En esta se utilizaron punteros, los cuales apuntaban a espacios de memoria almacenados en el heap, y se encontraban accesibles para cuando el proceso requiriera su uso.

Es posible que en algún momento de la ejecución de nuestra simulación más de un proceso quiera modificar el mismo espacio de memoria al mismo tiempo y de forma concurrente, pero esto no representaría problema alguno ya que se implementaron variables de tipo **atomic<int>,** la cual admite esta clase de operaciones.