P3. Comunicación y sincronización entre processos

Sistemas Operativos 1
Marzo 2025

1 Comunicación bidireccional entre procesos

Durante el desarrollo de esta práctica nos centraremos en utilizar algunos de los métodos que nos ofrece el sistema operativo para comunicar procesos entre sí. Existen múltiples métodos de comunicación entre procesos entre los que podemos mencionar las tuberías, los archivos, la red así como las señales.

En esta práctica se desarrollará un esquema de comunicación bidireccional entre procesos. En dicho esquema, un proceso se comunica con otro enviando o recibiendo información. La necesidad de que la información fluya de manera bidireccional tendrá un reflejo en la implementación práctica de la comunicación entre procesos.

2 Descripción de la práctica

Los ecosistemas naturales proveen recursos fundamentales para el bienestar humano, desde alimentos y agua hasta materias primas. Sin embargo, la sobreexplotación de estos recursos amenaza tanto su disponibilidad futura como la estabilidad de los ecosistemas que los sustentan. Esta realidad se refleja en varios Objetivos de Desarrollo Sostenible (ODS) de la ONU, particularmente en el ODS 12 (Producción y Consumo Responsables), el ODS 14 (Vida Submarina) y el ODS 15 (Vida de Ecosistemas Terrestres).

En esta práctica, exploraremos los fundamentos de la comunicación entre procesos en sistemas operativos a través de un problema de alta relevancia global: la gestión coordinada de recursos naturales compartidos. Implementaremos un sistema donde varios países extraen recursos de un ecosistema común, mientras que un agente coordinador monitoriza la salud del ecosistema y establece límites de extracción adaptables. El escenario propuesto tiene relación con una situación denominada, la "tragedia de los bienes comunales"[1] (del inglés, "*tragedy of the commons*"), un concepto clave en la gestión sostenible de recursos, donde actores individuales que priorizan su beneficio inmediato pueden degradar o incluso destruir un recurso compartido si no existe coordinación efectiva.

Más específicamente, el problema que se propone simula la gestión sostenible de un recurso natural genérico con una capacidad de carga máxima de 1 000 unidades (esta formulación abstracta permite aplicar el modelo a diversos recursos como poblaciones de peces, masas forestales o acuíferos, entre otros). La simulación se estructura en ciclos anuales donde interactúan tres actores principales: un agente coordinador y dos países. El coordinador mantiene el estado global del recurso, calcula su regeneración natural tras cada ciclo según el nivel actual, y establece límites máximos de extracción, que comunica a los países. Los países, por su parte, determinan su extracción dentro de estos límites establecidos y la comunican al coordinador. El coordinador actualiza entonces el estado del recurso, restando las extracciones y añadiendo un cierto grado de regeneración natural, siempre respetando la capacidad máxima del ecosistema. Este ciclo se repite año tras año, simulando la evolución del recurso bajo diferentes límites de extracción y ante condiciones variables de regeneración, que pueden incluir eventos aleatorios favorables o adversos que afectan la capacidad de recuperación del ecosistema.

En el sistema propuesto, el comportamiento de los países es sencillo, ya que únicamente generan sus demandas de recursos de forma aleatoria, como una cantidad de unidades entre

[1] Ver por ejemplo la entrada de Wikipedia: https://es.wikipedia.org/wiki/Tragedia_de_los_bienes_comunales

cero y el límite máximo establecido por el coordinador. Por el contrario, el establecimiento de los límites de extracción y la tasa de recuperación anual del recurso, son procesos algo más complejos y se describen con detalle en la Sección 3.

Con objeto de implementar el programa propuesto, utilizaremos tres procesos que incorporan la capacidad de comunicarse de manera bidireccional. Específicamente, definiremos un **proceso padre** que hará las veces de coordinador. Si nos encontramos en el año i de la simulación, el coordinador decidirá el límite de recursos que se pueden extraer en dicho año. Por otro lado, cada país será implementado por un **proceso hijo**, que se encargará de enviar al coordinador la cantidad de recursos que se solicita extraer en cada año al proceso padre. Además, el coordinador también debe descontar los recursos extraídos por cada país (siempre que existan recursos suficientes) y después de ello, calcular en qué medida se produce el crecimiento o reposición de los mismos de cara al siguiente año.

El programa a implementar ha de simular un total de `N` años, valor que se suministrará como parámetro de entrada. Para que el código pueda funcionar correctamente, será necesario definir un mecanismo que permita que el proceso padre notifique a los hijos el inicio de cada nuevo año. En este caso, el mecanismo de notificación (o sincronización) se implementará mediante **señales**, y más concretamente a través de la señal `SIGUSR1`. Por otro lado, también se necesitará un **búffer de comunicación** para que cada proceso hijo envíe al padre la cantidad de recursos extraídos en cada turno, y otro búffer más para que el padre envíe a los hijos el límite de extracción de recursos establecido. Se utilizará una **tubería sin nombre** para la comunicación de padre a hijo, y otra más para la de hijo a padre. Se proporciona un esquema de los principales elementos de la implementación solicitada en la Figura 1.

Cuando se hayan consumido los `N` años de la simulación, el proceso padre imprimirá un mensaje informando de las cantidades de recursos extraídas por cada país, la cantidad total extraída, y la cantidad de recursos que quedó disponible al final de todos los años.

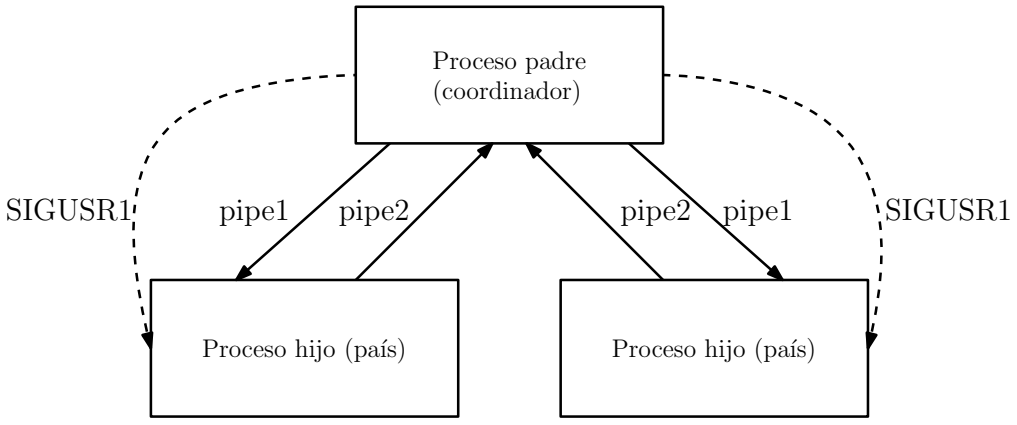

Figura 1: Esquema de comunicación bidireccional entre procesos a implementar. Hay un proceso padre (coordinador) y dos procesos hijos (países). La sincronización entre los processos se realiza mediante la señal `SIGUSR1`. Se usará una tubería sin nombre para que el coordinador envíe información a cada prisionero (`pipe1`) y otra tubería para que cada país envíe información al coordinador (`pipe2`).

3 Dinámica del ecosistema simulado

El modelo de simulación propuesto se basa en dos elementos fundamentales que reflejan la dinámica de los ecosistemas reales: por un lado, el establecimiento de límites de

extracción que varían según el estado del recurso, representando medidas de gestión adaptativa; y por otro, un mecanismo de recuperación que simula la regeneración natural del ecosistema bajo distintas condiciones. A continuación se detallan estos dos componentes que constituyen el núcleo del sistema a implementar y que determinarán la sostenibilidad a largo plazo del recurso compartido.

3.1 Umbrales de carga y límites de extracción de recursos

Se comienza la simulación con un total de 1 000 unidades de recursos, a lo que nos referimos como capacidad de carga del sistema. Los recursos totales no podrán ser superiores a dicha capacidad, ni tampoco inferiores a cero. Dentro del rango permitido, se definen tres umbrales de carga: el umbral alto (igual a la capacidad de carga), el umbral medio (igual a 750 unidades) y el umbral bajo (igual a 450 unidades).

Dependiendo de la cantidad actual de recursos y de los umbrales, el coordinador aplicará tres límites de extracción de recursos: limites alto, medio y bajo, según se muestra en la Tabla 1. Estos límites de extracción deben ser proporcionados por el usuario como parámetro de entrada del programa a implementar, a través de la línea de comandos.

Recursos Actuales	**Límite de Extracción**
$1000 \leq$ recursos < 750	límite alto
$750 \leq$ recursos < 450	límite medio
$450 \leq$ recursos ≤ 0	límite bajo

Tabla 1: Límites de extracción de recursos en función de los umbrales de carga.

3.2 Recuperación anual de los recursos

Una vez que el coordinador descuenta de los recursos actuales las extracciones llevadas a cabo por los dos países, cada año simulado finaliza calculando la recuperación anual de los recursos. Esta recuperación es un número que se suma a los recursos actuales, y que constituirá la cantidad de recursos con la que se comienza el siguiente año.

Se asume que la cantidad de unidades en que se recuperan los recursos es un determinado porcentaje, P, de los recursos actuales, siendo P mayor o igual que cero. Dicho porcentaje se compone a su vez de un porcentaje base, P_B, que depende de los umbrales de extracción, y una variación que se relaciona con la ocurrencia de eventos que afectan a la recuperación de recursos, P_V. Lo anterior se puede resumir con la ecuación siguiente:

$$P = \text{máx}\,(P_B + P_V, 0) \tag{1}$$

La Tabla 2 muestra cómo se determina P_B en función de los recursos actuales y los distintos umbrales. Los valores de la tabla intentan reflejar que la recuperación del ecosistema es óptima cuando los recursos se encuentran en un nivel intermedio (entre 450 y 750 unidades), mientras que disminuye tanto cuando el recurso está sobreexplotado (por debajo de 450 unidades) como cuando está cerca de su capacidad máxima (por encima de 750 unidades), simulando así el principio ecológico dependiente de la densidad observado en poblaciones naturales (baja densidad de individuos resulta en problemas para reproducirse, mientras que una densidad muy alta crea problemas de competencia por recursos limitados).

Por otro lado, la determinación de P_V se relaciona con la ocurrencia de 3 posibles eventos: que se produzcan condiciones normales, condiciones adversas o condiciones favorables de recuperación. Dicha ocurrencia se determina aleatoriamente. Una vez determinado cuál de los tres eventos ha tenido lugar, el valor de P_V queda definido según se muestra en la Tabla 3.

Recursos Actuales	P_B (%)
$1000 \leq$ recursos < 750	5
$750 \leq$ recursos < 450	20
$450 \leq$ recursos ≤ 0	5

Tabla 2: Determinación de P_B en función de los umbrales de carga.

Evento	Probabilidad	P_v (%)
Condiciones normales de recuperación	75	0
Condiciones adversas de recuperación	15	-15
Condiciones favorables de recuperación	10	10

Tabla 3: Determinación de P_V en función de tres posibles eventos.

Si usamos R_A para denotar los recursos actuales, e I para denotar el incremento de recursos, dicho incremento se calcularía de la siguiente manera:

$$I = R_A * \frac{P}{100} \tag{2}$$

Una vez obtenido el incremento, se obtendrían los recursos para el nuevo año, R_N, teniendo en cuenta que no pueden superar la capacidad de carga:

$$R_N = \text{mín}\,(R_A + I, 1\,000) \tag{3}$$

4 Algoritmo a implementar

En la Figura 1 puede verse el esquema de los procesos involucrados en el algoritmo a implementar. La implementación debe hacerse de manera que el valor de `N` sea conocido tanto por el padre como por los hijos. A continuación se detalla el algoritmo a implementar:

1. El programa lee los parámetros de entrada, a saber, los límites de extracción de recursos alto, medio y bajo, así como el número de turnos a ejecutar, `N`.

2. Se crean dos tuberías sin nombre mediante la función `pipe`. Una permitirá al proceso padre enviar información a los hijos (`pipe1`) y otra permitirá a los hijos enviar información al padre (`pipe2`).

3. Mediante `signal`, se registra la función para manejar la señal `SIGUSR1`.

4. Se crean los dos procesos hijo, para lo cual se hace uso de la función `fork`. Se recomienda poner el código del proceso hijo en una función. Dicha función ha de llevar a cabo la siguiente:

   a) Inicializar la semilla del generador de números aleatorios mediante `srand`, que espera como parámetro un número que debería ser específico del proceso hijo. Esta inicialización será útil a la hora de generar los valores de extracción anuales de cada país (ver más detalles en la Sección 5.4).

   b) Ejecutar un bucle de `N` iteraciones. En cada una de ellas es necesario:

      1) Esperar a que el proceso reciba la señal `SIGUSR1`.
      2) Una vez recibida la señal, usar la función `read` para leer desde `pipe1` el límite de extracción de recursos establecido por proceso padre.

3) A continuación, generar aleatoriamente una cantidad de recursos que se solicita extraer dentro del límite indicado por el proceso padre. Para más información sobre los distintos límites de extracción, puede consultarse la Sección 3.1.
4) Por último, la opción generada se ha de transmitir por la tubería `pipe2` mediante la función `write`.

5. Se ejecuta el código del proceso padre, para el que también se aconseja crear una función específica. El código a incorporar dentro de la función debería llevar a cabo los pasos siguientes:

   a) Ejecutar un bucle de `N` iteraciones. A continuación se muestran las acciones que deben llevarse a cabo en cada una de ellas:
   1) Se imprime un mensaje indicando que ha comenzado un nuevo año.
   2) Se indican los recursos actuales mediante un mensaje.
   3) Se calcula el límite de extracción de recursos para el año en curso y se muestra por pantalla (ver Sección 3.1 para obtener más información).
   4) Se envía la señal `SIGUSR1` de inicio de turno al país 1. Después se usa `write` sobre la tubería `pipe1` para escribir al país 1 el límite de extracción para el presente año. Una vez completada la escritura, se lee mediante `read` la cantidad de recursos solicitada por el país 1 a partir de la tubería `pipe2`. La cantidad de recursos solicitada, se resta de la cantidad de recursos actuales. Si la cantidad solicitada fuera mayor que los recursos disponibles, se deja a cero, no permitiéndose una cantidad de recursos negativa. Además, la extracción solicitada por el país 1 se acumula en una variable específica.
   5) El proceso anterior se repite para el prisionero 2.
   6) Una vez conocidas las extracciones para ambos países, se calcula la cantidad de unidades en que los recursos se recuperan dentro del año en curso y se muestra por pantalla (ver Sección 3.2 para más detalles). Se recomienda definir una función para calcular la cantidad en que los recursos se recuperan. Esta función recibe como entrada la cantidad de recursos actuales, y devuelve la cantidad en que dichos recursos se incrementan.

   b) Una vez acabadas las iteraciones, se ha de imprimir por pantalla la información sobre las extracciones acumuladas para cada país, la cantidad total de recursos extraída por ambos, y por último, la cantidad de recursos disponible al final de todos los años.

6. Antes de acabar el programa, se cierran los dos descriptores de fichero asociados a las dos tuberías creadas con `pipe`.

5 Ayuda para la implementación

Junto a esta práctica se entrega una plantilla (`practica3_plantilla.c`) a partir de la cual se puede empezar a trabajar. Esta plantilla incluye constantes útiles en el proceso de implementación, así como la lectura de los parámetros de entrada dentro de la función `main`.

El siguiente paso sería comenzar a añadir el código al final de la función `main` de manera que se implemente el mecanismo de asignación de recursos que se propone para la práctica. Dicha código requiere la comunicación bidireccional entre procesos tal y como se ha descrito anteriormente (ver Figura 1).

Se indican a continuación diversos aspectos a tener en cuenta a la hora de implementar la práctica.

5.1 Ejemplo de salida del programa

A continuación se muestra un ejemplo de la salida que deberá producir el programa para un total de 4 años de simulación. Cada vez que empieza un año, se muestra su número, y después una serie de entradas, todas impresas desde el proceso coordinador. Estas entradas proporcionan información del desarrollo de cada año. Cada entrada lleva un prefijo indicando su origen. Cada información relacionada con la asignación de recursos llevará `[Coordinador]` como prefijo. Por otro lado, aquella información relacionada con eventos que afectan a la recuperación de los recursos, vendrá precedida por la cadena `[Evento]`.

```
* AÑO 1
[Coordinador] Los recursos disponibles para el año en curso son 1000.000000
[Coordinador] El límite de extracción para el año en curso es 70.000000
[Coordinador] El pais 1 solicita extraer 2.156183
[Coordinador] La solicitud de extracción del país 1 se ha aprobado
[Coordinador] El pais 2 solicita extraer 61.910748
[Coordinador] La solicitud de extracción del país 2 se ha aprobado
[Evento] Este año se han producido condiciones normales de recuperación
[Evento] El porcentaje de recuperación es del 5.000000%
[Coordinador] Los recursos se recuperaron en 46.796654 unidades

* AÑO 2
[Coordinador] Los recursos disponibles para el año en curso son 982.729675
[Coordinador] El límite de extracción para el año en curso es 70.000000
[Coordinador] El pais 1 solicita extraer 15.671197
[Coordinador] La solicitud de extracción del país 1 se ha aprobado
[Coordinador] El pais 2 solicita extraer 9.286984
[Coordinador] La solicitud de extracción del país 2 se ha aprobado
[Evento] Este año se han producido condiciones normales de recuperación
[Evento] El porcentaje de recuperación es del 5.000000%
[Coordinador] Los recursos se recuperaron en 47.888577 unidades

* AÑO 3
[Coordinador] Los recursos disponibles para el año en curso son 1000.000000
[Coordinador] El límite de extracción para el año en curso es 70.000000
[Coordinador] El pais 1 solicita extraer 20.809481
[Coordinador] La solicitud de extracción del país 1 se ha aprobado
[Coordinador] El pais 2 solicita extraer 41.772758
[Coordinador] La solicitud de extracción del país 2 se ha aprobado
[Evento] Este año se han producido condiciones normales de recuperación
[Evento] El porcentaje de recuperación es del 5.000000%
[Coordinador] Los recursos se recuperaron en 46.870888 unidades

* AÑO 4
[Coordinador] Los recursos disponibles para el año en curso son 984.288635
[Coordinador] El límite de extracción para el año en curso es 70.000000
[Coordinador] El pais 1 solicita extraer 46.868603
[Coordinador] La solicitud de extracción del país 1 se ha aprobado
[Coordinador] El pais 2 solicita extraer 68.889755
[Coordinador] La solicitud de extracción del país 2 se ha aprobado
[Evento] Este año se han producido condiciones adversas de recuperación
[Evento] El porcentaje de recuperación es del 0.000000%
[Coordinador] Los recursos se recuperaron en 0.000000 unidades

La simulación ha finalizado.
Recursos extraídos por país 1: 85.505463
Recursos extraídos por país 2: 181.860245
Total recursos extraídos: 267.365723
Recursos disponibles: 868.530273
```

5.2 Sincronización mediante señales

La sincronización entre procesos se realizará mediante la señal `SIGUSR1`, que el proceso padre enviará a los hijos mediante la función `kill`. Recuérdese que el padre envía la señal a los hijos para indicarles el comienzo de un nuevo año y que deben, por un lado leer del padre el límite de extracción de recursos para dicho año (`pipe1`), y por otro lado, llevar a

cabo una solicitud de extracción en base a ese límite (`pipe2`).

Al realizar pruebas con la implementación del código propuesto pueden encontrarse problemas si el mecanismo de sincronización entre procesos utiliza la función `pause`, que bloquea un proceso hasta que recibe y atiende una señal. La razón es que existe la posibilidad de que la señal sea recibida y atendida antes de que se ejecute `pause`, lo que causaría que el proceso quedase bloqueado. Una posible solución es utilizar la denominada *espera activa* (en inglés *busy waiting* o *spinning*) para gestionar la sincronización entre los procesos hijo y el padre en lugar de utilizar la función `pause`. La espera activa involucra un código similar al que se muestra a continuación:

```
while (!sigusr1) {};
sigusr1 = 0;
```

En el ejemplo anterior, el código espera consumiendo ciclos de CPU y sin bloquearse, con objeto de recibir la señal `SIGUSR1`. Supongamos que `sigusr1` es una variable global inicializada a 0, y que al recibir la señal la función que gestiona las señales pone `sigusr1` a 1. En el código propuesto no se hace ninguna llamada a `pause`: al recibir la señal se pone la variable `sigusr1` a 1 y el código sale del bucle `while` para volverla a poner a 0 de forma inmediata. Obsérvese que el código funcionará aunque se reciba (y se procese) la señal antes de entrar a la espera activa.

La solución propuesta puede ser aceptable en este caso, dado que sabemos que los procesos hijo no tendrán que esperar mucho para recibir la señal por parte del padre. Por tanto, no se desperdiciará demasiado tiempo de CPU[2].

5.3 Transferencia de datos bidireccional

En esta sección se describe cómo implementar la transferencia de datos desde entre el proceso padre y los procesos hijos en cada año de la simulación, según se mencionaba en el algoritmo descrito en la Sección 4.

Específicamente, cada proceso hijo será tratado por el proceso padre de forma secuencial. Empezando por el primer hijo, el padre le enviará la señal `SIGUSR1` que indica al hijo el inicio de un nuevo año. En dicho año, el hijo debe recibir del padre el límite de extracción de recursos para el año actual. A continuación, el hijo debe enviar la solicitud de extracción de recursos. El proceso padre leerá la solicitud y repetirá el proceso anterior para el segundo hijo, concluyendo el año.

El envío por parte del padre a los hijos del límite de extracción de recursos para el año en curso se llevará a cabo mediante una tubería sin nombre (`pipe1`) definida mediante la función `pipe`. A `pipe` se le pasará como parámetro un array de dos posiciones, que tras la ejecución contendrá en la primera posición el descriptor que se utiliza para lectura, y en la segunda posición el que se usará para escritura. El descriptor de lectura será usado por el hijo, quien podrá leer el límite de extracción de recursos por medio de la función `read`. Por otro lado, el descriptor de escritura será utilizado por el padre para enviar el límite de extracción de recursos mediante la función `write`.

El envío al proceso padre de las solicitudes de extracción de recursos se llevará a cabo también mediante una tubería sin nombre (`pipe2`), pudiéndose hacer sobre ella consideraciones análogas a las que se hicieron más arriba sobre la tubería usada por el padre para enviar información a los procesos hijo.

[2]No obstante, la sincronización también puede hacerse sin usar espera activa, según se propone en el ejercicio adicional de esta práctica descrito en la Sección 6.2

5.4 Toma de decisiones de los países

En esta sección se proporciona ayuda para implementar la toma de decisiones en los países. Para este propósito, es necesario generar aleatoriamente la cantidad de recursos a extraer.

Dicha generación aleatoria puede parecer trivial a simple vista, ya que tan sólo es necesario generar aleatoriamente un número entre cero y el límite de extracción de recursos impuesto por el coordinador. Este objetivo se puede conseguir por medio de la función `rand`[3]. No obstante, si la semilla de generación de números aleatorios no se establece apropiadamente por medio de la función `srand`, los dos procesos hijo generarían siempre la misma demanda de extracción, ya que ambos parten de copias idénticas del espacio de memoria del proceso padre.

Típicamente, a `srand` se le proporciona como parámetro de entrada el tiempo del sistema, que se obtiene mediante la llamada `time(NULL)`. En particular, la llamada a `time` devuelve la cantidad de segundos transcurrida desde una fecha determinada considerada como referencia hasta el momento actual. Sin embargo, en nuestro caso esta estrategia no sería válida, ya que ambos procesos hijo se crean prácticamente a la vez, lo que desemboca en una semila idéntica para los dos.

Cualquier solución al problema que se acaba de exponer, pasa por inicializar la semilla de números aleatorios con un número que dependa del proceso hijo. Una posible vía para conseguirlo sería basarse en la función `getpid`, que devuelve un identificador numérico del proceso que la ejecuta.

5.5 Depuración del código

Es habitual utilizar las llamadas a `printf` para depurar el código. Aunque puede ser útil en algunos casos, es conveniente mencionar que la introducción de estas instrucciones puede hacer que cambien los puntos en que los procesos hijo y el padre hagan un cambio de contexto. Por tanto, partiendo de una sincronización erróneamente implementada, es posible que añadiendo instrucciones `printf` el código funcione correctamente mientras que eliminándolas no lo haga. Será necesario por tanto analizar con detalle el código para encontrar la fuente del problema.

6 Ejercicios opcionales

Se proponen dos ejercicios opcionales para complementar el trabajo que se acaba de proponer. El primero consiste en automatizar la realización de experimentos para distintos valores de los parámetros de entrada mediante Bash. Con respecto al segundo, se propone modificar del sistema de sincronización entre el proceso padre y los dos procesos hijo de forma que se evite la espera activa. Si se implementa con éxito este último, se podrá subir la nota de la entrega hasta 1 punto.

6.1 Experimentación con barrido de parámetros mediante Bash

Dado que los límites de extracción se pueden definir de múltiples formas, resulta de interés llevar a cabo una experimentación que haga un barrido de parámetros, recogiendo la cantidad de recursos extraídos a los largo de una serie de años. Para ello, se propone crear un shell script de Bash que ejecute el algoritmo siguiente:

[3] NOTA: `rand` genera aleatoriamente números enteros entre 0 y una constante llamada `RAND_MAX`, por lo que el número generado debe ser escalado.

1. Definir un triple bucle `for` anidado en el que se recorra un rango de valores para cada uno de los tres límites de extracción. Por ejemplo, se podría iterar tomando valores de 0 a 200 para cada límite en incrementos de 20.

2. En cada iteración del bucle más interno, se llevarían a cabo las siguientes tareas:

   a) Llamar al ejecutable de la práctica que se ha creado, para los límites correspondientes a la iteración en curso, y para una cantidad suficiente de años, por ejemplo, se puede establecer $N = 1\,000$.

   b) Redirigir la salida del experimento anterior a un fichero.

   c) A partir del fichero, obtener la cantidad de recursos total extraída para cada experimento en una variable.

   d) Mostrar por pantalla, en una única línea, las siguientes 4 columnas: límite de extracción alto, límite de extracción medio, límite de extracción bajo y cantidad de recursos total extraída.

La salida por pantalla generada por el script de automatización que se ha descrito, se puede redirigir a un fichero. Una vez hecho esto, se puede implementar un script adicional que tome ese fichero y devuelva los límites que corresponden a la mayor cantidad de recursos extraída.

Adicionalmente, y usando los conocimientos de la asignatura de "Introducción a la Computación Científica", se pueden generar representaciones gráficas de los resultados mediante la herramienta `gnuplot`. Una posibilidad sería, por ejemplo, tomar todos los experimentos con el límite de extracción alto igual a 100 (u otro que se considere de interés), y generar una gráfica en 3D, tomando como variables independientes los límites medio y bajo establecidos, y como variable dependiente, la cantidad de recursos extraídos.

6.2 Sincronización sin espera activa

Tal como se explicaba en la Sección 5.2, la implementación que se propone utiliza espera activa en lugar de la función `pause` con objeto de sincronizar el proceso padre con los dos procesos hijo. La implementación por medio de `pause` sería incorrecta debido a que existe la posibilidad de que dicha función se llame después de que la señal `SIGUSR1` haya sido recibida y atendida, lo que desembocaría en un bloqueo del proceso hijo.

A pesar de que, como se explicaba más arriba, la espera activa no tiene un impacto demasiado negativo en esta implementación, es algo que debería evitarse siempre que sea posible. Se propone como ejercicio adicional, proporcionar una implementación de la sincronización sin espera activa. Dicha implementación se basará en lo siguiente:

- Existe la posibilidad de bloquear señales, de manera que éstas quedan en espera para ser atendidas. El bloqueo se puede llevar a cabo de forma específica por cada señal. Cuando las señales se desbloquean, se atienden en el orden en que fueron recibidas. A fin de bloquear señales, podemos hacer uso de la función `sigprocmask`.

- La función `sigprocmask` trabaja con variables que representan conjuntos de señales que están bloqueadas. A estos conjuntos de señales se les llama máscaras, o en inglés *masks*. Las máscaras de señales se representan con un tipo específico llamado `sigset_t`. Entender el concepto de máscara de señales es fundamental en la nueva implementación que se propone.

- Para evitar el problema que se ha mencionado con `pause`, es necesario añadir la señal `SIGUSR1` a la máscara de señales bloqueadas mediante `sigprocmask` antes de que la señal pueda enviarse. Por tanto, debería usarse `sigprocmask` dentro de la función

`main` antes siquiera de crear los procesos hijo mediante `fork`. Se recomienda guardar, tanto la máscara inicial que había antes de llamar a `sigprocmask` como la nueva máscara que también incluye `SIGUSR1` en dos variables globales.

- En los procesos hijo debe quedar bloqueada `SIGUSR1`, mientras que en el proceso padre podemos desbloquearla (nuevamente usando `sigprocmask`) antes de ejecutar el resto del código que ya habíamos implementado (en realidad, no sería estrictamente necesario desbloquearla porque el proceso padre no recibirá estas señales, sólo las enviará, no obstante el código resulta más elegante si se desbloquea).

- Una vez hecho lo anterior, podemos dejar vacío el código de la función que trata la señal `SIGUSR1`, ya que no usaremos espera activa. Por otro lado, el bucle que implementa esta espera activa en el proceso hijo ha de sustituirse por una llamada a la función `sigsuspend`. La función `sigsuspend` ejecuta tres acciones. En primer lugar, establece temporalmente una máscara indicando qué señales están bloqueadas. Una vez establecida la nueva máscara se ejecuta internamente la función `pause`, lo que provoca que, si como resultado del establecimiento de la nueva máscara se desbloquean señales que estaban pendientes de ser atendidas, el proceso detendrá su ejecución hasta que se atienda una de ellas. En tercer y último lugar, la anterior máscara de señales se reestablece y `sigsuspend` termina.

- En nuestro caso, `sigsuspend` ha de establecer una máscara de señales que no bloquea `SIGUSR1` (se puede usar la máscara antigua que devuelve la primera llamada a `sigprocmask`). Esto permite que `sigsuspend` ejecute internamente `pause` para esperar a que se capture `SIGUSR1` sin ningún peligro de que esta señal ya se hubiera recibido antes, ya que ha permanecido todo el tiempo bloqueada. Además, después de ejecutar `pause` internamente, la máscara anterior se restablece, por lo que `SIGUSR1` vuelve a estar bloqueada.

Como ayuda adicional para la nueva implementación, son útiles otras dos funciones que se relacionan con el trabajo con máscaras de señales:

- Es posible crear una máscara de señales vacía mediante la función `sigemptyset`.

- Dada una máscara de señales, podemos añadir una nueva señal mediante la función `sigaddset`.

Se recomienda hacer uso del comando `man` para visualizar información sobre las distintas funciones que serán necesarias para la nueva implementación.

7 Entrega

La entrega consiste en dos partes: **el código fuente** (80 % de puntuación) y **el informe** (20 % de puntuación).

7.1 Código fuente

Se pide entregar un único fichero en C implementando la funcionalidad descrita en la Sección 4. Una vez compilado el fichero de código fuente, el programa resultante tendrá cuatro argumentos: los límite de extracción con nivel de recursos alto, medio y bajo, así como el número de turnos, N, que se jugarán.

```
$ ./practica3 <limite_extr_alto> <limite_extr_medio> <limite_extr_bajo> <valor_N>
```

Si se completa el ejercicio adicional automatizando la realización de experimentos, se deberán adjuntar también el fichero o ficheros con los shell scripts en Bash adicionales (ver formato de la entrega en la Sección 7.3).

Además, si también se lleva a cabo el ejercicio adicional consistente en la implementación sin espera activa, deberá adjuntarse en un fichero aparte.

7.2 Informe

El informe a entregar debe prepararse en **formato PDF o equivalente (no se admiten formatos como odt, docx,...)**. Una forma adecuada de estructurar el informe para esta práctica sería definir tres secciones:

1. **Introducción**: se debe describir brevemente el problema que se ha solucionado (es decir, un resumen de lo que propone esta práctica).

2. **Implementación y pruebas**: por un lado, se pide describir cómo se ha estructurado el código, enumerando las distintas funciones y proporcionando una breve descripción del cometido de cada una de ellas. También se puede destacar la definición de alguna variable que se considere importante. Por otro lado, también se han de mostrar las pruebas que se han realizado para asegurar el buen funcionamiento del código. Si se ha implementado la automatización de la experimentación mediante Bash (ver Sección 6.1), se debe añadir un apartado extra describiendo brevemente el código y los resultados obtenidos.

3. **Conclusiones**: se debe hacer una breve reflexión sobre lo aprendido. De manera opcional, también se puede usar esta sección para aportar una reflexión personal sobre la práctica, incluyendo sugerencias sobre aspectos que se podían haber incluido a fin de hacerla más provechosa.

En el caso de que se desee incluir capturas de pantalla en lugar de incluir los resultados de los experimentos en formato texto, es necesario asegurarse de que el texto de la captura se puede leer bien (es decir, que tenga un tamaño similar al resto del texto del documento) y que todas las capturas sean uniformes (es decir, que todas las capturas tengan el mismo tamaño de texto).

El documento debe tener una longitud máxima de 4 páginas (sin incluir la portada). El documento se evaluará con los siguientes pesos: pruebas realizadas y comentarios asociados, un 60 %; escritura sin faltas de ortografía y/o expresión, un 20 %; paginación del documento hecha de forma limpia y uniforme, 20 %.

7.3 Formato de la entrega

Cada grupo debe subir **un único fichero ZIP** a la tarea asociada a la práctica en el Campus Virtual, con el nombre **P3_GrupoXX_nombre1_apellido1_nombre2_apellido2.zip**. El fichero debe incluir lo siguiente:

- Fichero `practica3.c` implementando el sistema descrito en este documento utilizando espera activa para la sincronización.
- Informe de la práctica en formato `pdf`.
- Fichero `practica3_sigsuspend.c` con la implementación del sistema sin usar espera activa (**OPCIONAL**).