Trabajo práctico n. 1

ConcuBread

7559 - Técnicas de programación concurrente I

Alumno: Rozanec, Matías padrón: 97404

Profesor: Deymonaz, Pablo

Facultad de Ingeniería - Universidad de Buenos Aires
1c2020

Índice

Objetivo

El objetivo de este proyecto consiste en simular el proceso de elaboración y atención de pedidos en un local de panificados para poder aplicar los conceptos de concurrencia, concretamente de IPC, estudiados en clase.

Análisis del problema

El problema presentado consiste en leer un archivo de configuración que indica, en última instancia, la cantidad de procesos hijos en roles de Recepcionistas, Panaderos y Pizzeros que deberá crear el padre.

Dicho archivo de configuración se debe presentar bajo el nombre de config.cb en el directorio raíz, con 3 líneas, cada una con un número indicando, en este orden:

- cantidad de panaderos
- cantidad de pizzeros
- cantidad de recepcionistas

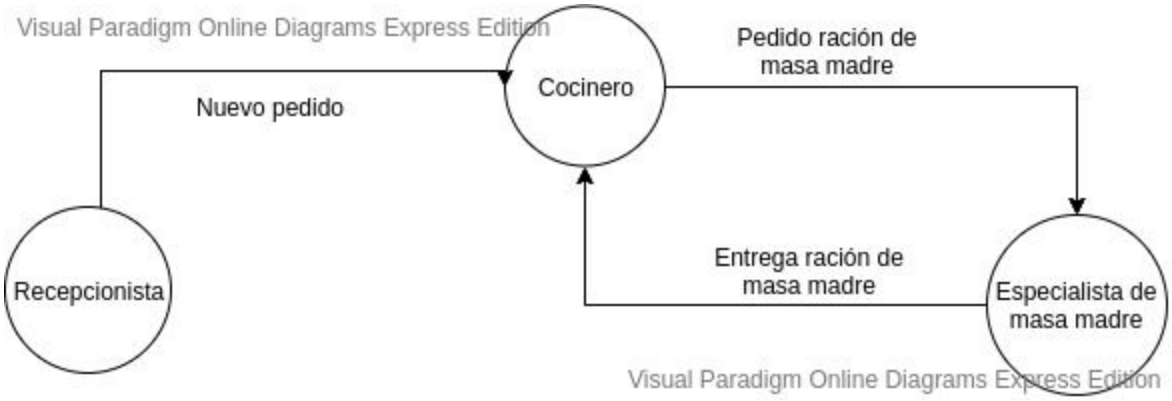
Además de estos procesos que son de cantidad variable, el proceso padre deberá crear otros procesos que completan la simulación. La lista de todos los procesos a crear para una simulación es la siguiente:

- Recepcionistas
- Panaderos
- Pizzeros
- Especialista de masa madre
- Debug printer

Además de los procesos nombrados existe naturalmente el proceso padre que se omite en la lista, porque es quien da vida al resto de los procesos.

El próximo paso después de detectar los procesos conformantes del problema es decidir qué procesos necesitan comunicarse entre sí y el tipo de IPC a utilizar en cada uno de los casos.

En el diagrama a continuación se presentan las principales comunicaciones que se tienen que dar entre los principales actores de la simulación. Nótese que en lugar de pensar en Panadero y Pizzero de forma individual, se los engloba a ambos en el papel de Cocinero.



Además de lo presentado en el diagrama, hay que notar que todos los procesos, incluyendo al padre, se comunican con el Debug printer para poder enviar mensajes de debug como pide el enunciado. Esto último no se incluyó en el diagrama porque entorpecería bastante el entendimiento del mismo.

Implementación de IPC

Los métodos estudiados hasta ahora son:

- Memoria compartida
- Señales
- Pipes y fifos
- Locks

Señales

En el tp se hace uso de manejo de señales con el objetivo de terminar de forma controlada el programa en caso de recibir una señal de tipo SIGKILL. No se les ha dado un uso más extensivo.

Memoria compartida vs Pipes

Ante la decisión de si usar memoria compartida o pipes (tanto pipes como named pipes - fifos), se decidió utilizar pipes por resultar de una naturaleza más simple.

Entre pipes y fifos se decidió utilizar fifos por su flexibilidad adicional que posibilita comunicación 1:N en los casos en que esto es requerido.

Locks

Finalmente, en un caso particular se usaron también Locks. Se trata del caso de los cocineros, donde múltiples cocineros (cantidad desconocida) deben recibir pedidos de una cantidad también desconocida de recepcionistas. Si bien las escrituras son atómicas, lo que posibilita que múltiples recepcionistas escriban en un mismo fifo sin problema, la lectura no lo es. Por lo tanto, en la lectura de los pedidos, se hace uso del lock para que no suceda que dos procesos quieran leer al mismo tiempo de un fifo.

Protocolos

A continuación se listan los protocolos utilizados en la comunicación entre partes. A excepción de los mensajes de debug, el resto de los mensajes tienen longitud fija para que los lectores sepan de a cuántos bytes deben leer.

Recepcionista

Cuando un recepcionista encarga un pedido, le asigna a éste un id único que permitiŕa seguir el pedido a lo largo de toda su vida. Este id se conforma del prefijo "Piz" o "Pan", según se trate de pizza o pan respectivamente, seguido por 4 números, donde los primeros dos indican el id del recepcionista, y los segundos dos el número de pizza o pan que dicho recepcionista esté recibiendo. Así, los primeros dos pizzas y panes de los primeros dos recepcionistas tendrán los siguientes ids:

- Piz0000 Recepcionista 0, primer pizza que encarga.
- Piz0001 Recepcionista 0, segunda pizza que encarga.
- Pan0000 Recepcionista 0, primer pan que encarga.
- Pan0001 Recepcionista 0, segundo pan que encarga.
- Piz0100 Recepcionista 1, primer pizza que encarga.
- Piz0101 Recepcionista 1, segunda pizza que encarga.
- Pan0100 Recepcionista 1, primer pan que encarga.
- Pan0101 Recepcionista 1, segundo pan que encarga.

Cocineros

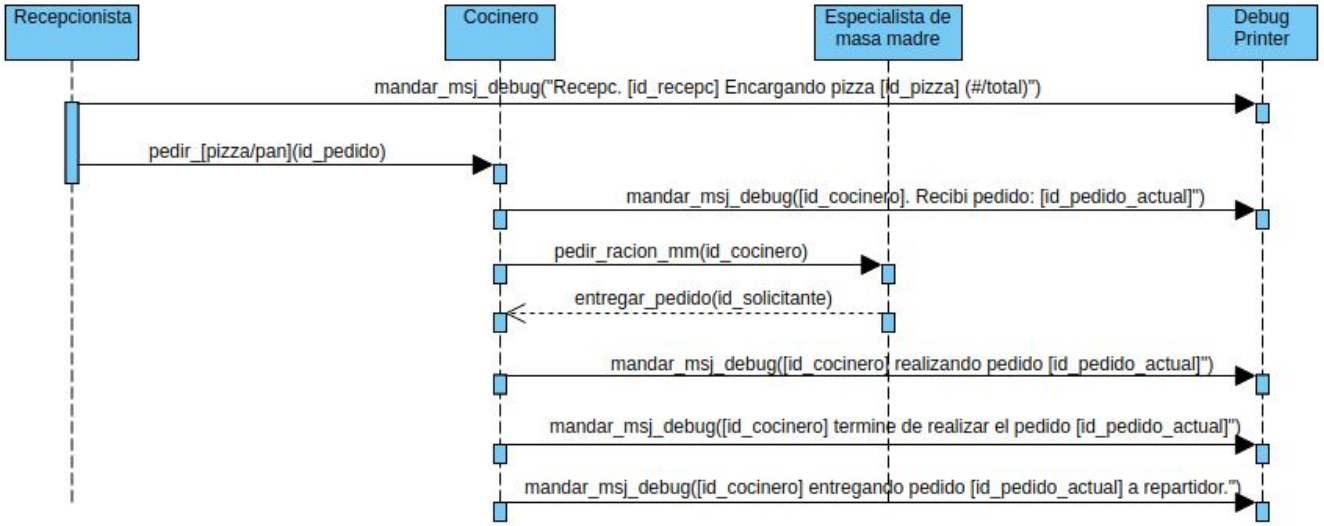
Los cocineros deben pedir las raciones de masa madre. Como el Especialista debe luego enviar la ración solicitada, es importante que sepa quién le está pidiendo dicha ración. Como además en todos los casos se pide una ración, sin especificar nada más, como los pedidos son siempre por un solo pan o una sola pizza y no se especifican tamaños ni cantidades de ración variables, se supone que todas las raciones a enviar son del mismo tamaño. Esto permite que la petición se produzca enviando solamente el id del cocinero.

Especialista de masa madre

Una vez realizado el pedido de masa madre, el especialista se encarga de cumplir con el pedido. Como es lo único que envía el especialista, lo hace enviando un mensaje de un solo byte al cocinero correspondiente.

Diagrama

En el diagrama de secuencia presentado a continuación se puede observar el flujo de los mensajes que se envían entre los distintos procesos, con toda la información correspondiente que permite identificar de forma unívoca a cada participante: recepcionista, cocinero y pedido. Se agrega el Debug printer porque es en definitiva el proceso que recibe toda la información que hace el seguimiento de un pedido, según indicación del enunciado. Esto hace al diagrama más explícito y por ende más fácil de comprender el funcionamiento del sistema completo.



Casos de uso

A continuación se listan funcionalidades de la aplicación a través de casos de uso correspondientes a funcionalidades desde la perspectiva de un usuario de la aplicación.

US - Simulación de cadena de pedido
Como usuario se quiere simular la cadena de producción de pan y/o pizza, pasando por recepcionistas, cocineros, especialista de masa madre y repartidor.
Criterios de aceptación
● Se puede especificar la cantidad de cocineros, panaderos y recepcionistas a usar en la simulación.

REC - Simulación de pedidos entrantes
Como recepcionista se quiere simular la entrada de pedidos, sean de pizza o de pan, y la consecuente delegación de los mismos hacia los cocineros.
Criterios de aceptación
● Se simula una cantidad de pedidos finita y aleatoria en orden aleatorio, sin especificar a qué cocinero en particular le corresponde cada pedido.

COCINERO - Simulación de pedidos entrantes
Como cocinero se quiere simular la entrada de pedidos y el proceso de creación del pedido, con las comunicaciones necesarias hacia otros procesos.
Criterios de aceptación
● Se simula la elaboración total del pedido, pidiendo ración de masa madre y entregando finalmente el pedido al repartidor.

ESPEC - Simulación de pedidos de masa madre
Como especialista de masa madre se quiere simular la entrada de pedidos de ración de masa madre y la entrega correspondiente.
Criterios de aceptación
● Se simula la escucha de pedidos durante la vida del programa hasta que finalmente no haya más pedidos y se cierren los canales de comunicación.

Diagrama de estados

A continuación se presenta un diagrama con todos los estados por los que pasa un pedido. Como se puede observar, todo pedido pasa por cuatro estados: estos son los cuatro estados que habrá que buscar en el log file por cada pedido para asegurarse que el pedido ha cumplido correctamente con su ciclo de vida.

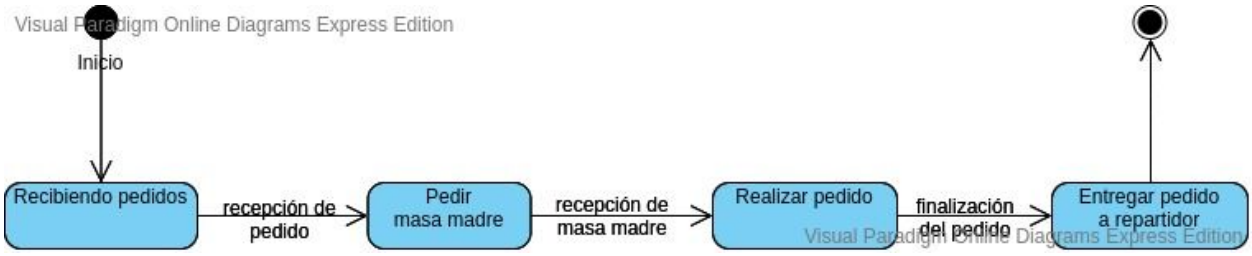
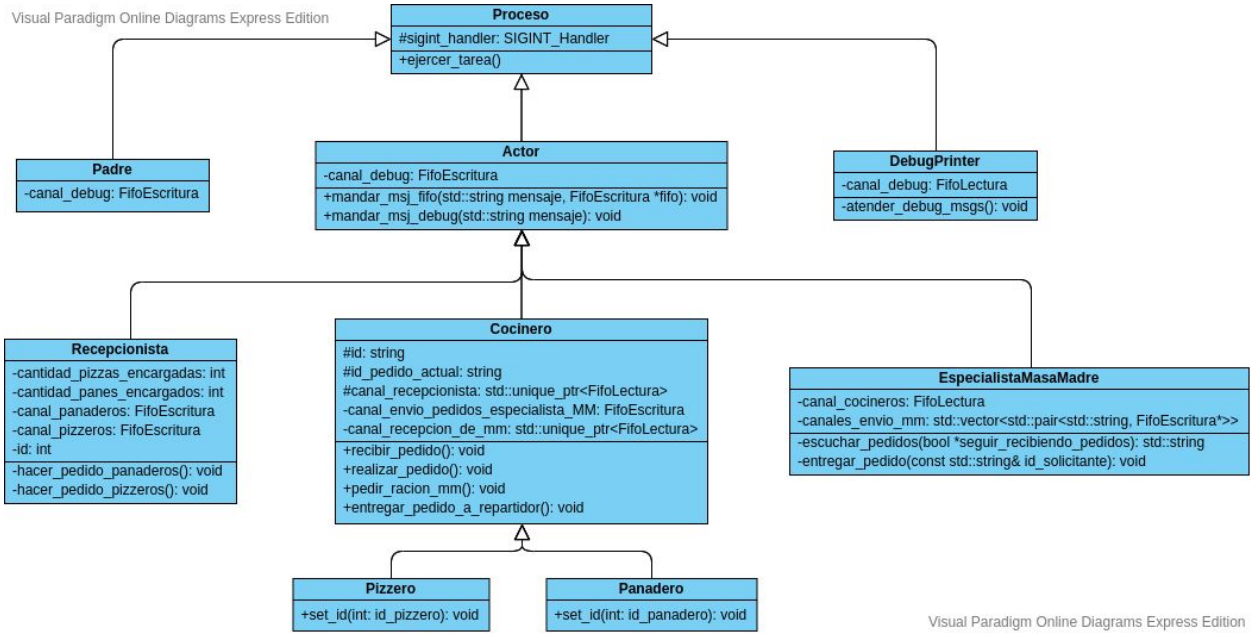


Diagrama de clases

A continuación se presenta el diagrama de clases. El diseño del trabajo práctico se vio muy simplificado a partir de la decisión de trabajar con una clase Proceso. Esto se puede observar más que nada en el main, donde se realizan los forks para crear todos los procesos necesarios. Mencionada tarea es delegada al Process Manager, que se encarga de devolver un unique pointer a Proceso. Dicho proceso luego ejecuta el método ejercer_tarea(), y al finalizar retorna 0. Queda así el main reducido a unas poquísimas líneas de código que manejan todos los casos posibles para cualquier tipo de proceso que se cree.

De la clase Proceso se puede ver que heredan tres otras clases, a saber: Padre, Actor y Debug Printer. Tanto el Padre como el Debug Printer cumplen funciones únicas que escapan al escenario simulado propiamente dicho. Por otro lado, todos los actores participantes de la simulación heredan de la clase Actor, que implementa los métodos de comunicación de FIFOs.

A partir de la clase Actor se distinguen tres grandes actores: Recepcionistas, Cocineros y Especialista en masa madre. Cabe destacar que hasta aquí llega el máximo nivel de complejidad, ya que la distinción entre Panaderos y Pizzeros es realmente ínfima: toda la implementación de todas las funcionalidades de ellos quedan desarrolladas en su totalidad en la clase Cocinero, siendo la única función de estas clases hijas la de la asignación de un identificador único.


Visual Paradigm Online Diagrams Express Edition
Proceso
#sigint_handler: SIGINT_Handler
+ejercer_tarea()
Padre
-canal_debug: FifoEscritura
Actor
-canal_debug: FifoEscritura
+mandar_msj_fifo(std::string mensaje, FifoEscritura *fifo): void
+mandar_msj_debug(std::string mensaje): void
DebugPrinter
-canal_debug: FifoLectura
-atender_debug_msgs(): void
Recepcionista
-cantidad_pizzas_encargadas: int
-cantidad_panes_encargados: int
-canal_panaderos: FifoEscritura
-canal_pizzeros: FifoEscritura
-id: int
-hacer_pedido_panaderos(): void
-hacer_pedido_pizzeros(): void
Cocinero
#id: string
#id_pedido_actual: string
#canal_recepcionista: std::unique_ptr<FifoLectura>
-canal_envio_pedidos_especialista_MM: FifoEscritura
-canal_recepcion_de_mm: std::unique_ptr<FifoLectura>
+recibir_pedido(): void
+realizar_pedido(): void
+pedir_racion_mm(): void
+entregar_pedido_a_repartidor(): void
EspecialistaMasaMadre
-canal_cocineros: FifoLectura
-canales_envio_mm: std::vector<std::pair<std::string, FifoEscritura*>>
-escuchar_pedidos(bool *seguir_recibiendo_pedidos): std::string
-entregar_pedido(const std::string& id_solicitante): void
Pizzero
+set_id(int: id_pizzero): void
Panadero
+set_id(int: id_panadero): void
Visual Paradigm Online Diagrams Express Edition

Código

Comentarios generales

Durante todo el desarrollo se hizo uso de herramientas del lenguaje c++11, entre las que se destaca el uso de smart pointers. Esto hace que exista un solo puntero a un objeto, haciendo fácil poder liberar la memoria en cuanto el smart pointer salga de scope, cosa que sucede automáticamente.

Se ha tenido especial cuidado de cerrar todos los fifos utilizados y de destruirlos en por lo menos uno de los extremos para que no haya leaks de memoria.

Fifos dinámicos

En la comunicación entre especialista de masa madre y cocineros, la forma en que el especialista le pueda enviar un mensaje a un cocinero en particular es mediante un canal que exista únicamente entre el especialista y el cocinero en cuestión. De lo contrario, de haber un canal solo donde en que escucharan todos los cocineros, no se podría controlar de antemano qué cocinero leerá cada mensaje, y por más que se incluya en el mensaje para quién es la ración de masa madre enviada, no serviría de mucho, ya que una vez que lo leyó un cocinero, no lo podrá volver a leer otro. Por este motivo, se crean fifos de forma dinámica, con el nombre de archivo igual al id del cocinero al que se debe enviar la ración de masa madre.

Implementación de tiempos y cantidades variables

La naturaleza de tiempos variables solicitada en el enunciado fue lograda mediante el uso de std::uniform_int_distribution. Además de los tiempos variables, se usó también para lograr una cantidad de encargos variables tanto de pan como de pizza para cada recepcionista en particular.

Limitaciones

La cantidad de cocineros y recepcionistas no debe exceder 100 en cada uno de los casos. Es decir, podrá haber como máximo 100 recepcionistas, 100 panaderos y 100 pizzeros. Esto se debe a la asignación de ids.

Anexo

Código

El código se puede ver en el siguiente repositorio de Github
https://github.com/rozanecm/7559_ConcuBread
Además, en las últimas páginas de este informe se anexa el código para poder leerlo más fácilmente.

Estructura del código

A continuación se muestran los contenidos y directorios pertenecientes al código en forma de árbol para una más fácil navegación en el mismo.

```
src/
├─── fifos
│    ├─── Fifo.cpp
│    ├─── FifoEscritura.cpp
│    ├─── FifoEscritura.h
│    ├─── Fifo.h
│    ├─── FifoLectura.cpp
│    └─── FifoLectura.h
├─── main.cpp
├─── Procesos
│    ├─── Actores
│    │    ├─── Actor.cpp
│    │    ├─── Actor.h
│    │    ├─── Cocinero.cpp
│    │    ├─── Cocinero.h
│    │    ├─── EspecialistaMasaMadre.cpp
│    │    ├─── EspecialistaMasaMadre.h
│    │    ├─── Panadero.cpp
│    │    ├─── Panadero.h
│    │    ├─── Pizzero.cpp
│    │    ├─── Pizzero.h
│    │    ├─── Recepcionista.cpp
│    │    └─── Recepcionista.h
│    ├─── DebugPrinter.cpp
│    ├─── DebugPrinter.h
│    ├─── Padre.cpp
│    ├─── Padre.h
│    ├─── Proceso.cpp
│    └─── Proceso.h
```

```
├──── signals
│     ├──── EventHandler.h
│     ├──── SIGINT_Handler.h
│     ├──── SignalHandler.cpp
│     └──── SignalHandler.h
└──── utils
      ├──── consts.h
      ├──── ProcessManager.cpp
      └──── ProcessManager.h
```

Enunciado

En las siguientes páginas y antes del código se anexa el enunciado del trabajo.

Primer Proyecto: ConcuBread

75.59 - Técnicas de Programación Concurrente I

Objetivo

Las circunstancias actuales del mundo llevaron a un cambio de hábitos: aumentó el consumo de pan y la forma de compra por delivery.

El objetivo de este proyecto consiste en simular el proceso de elaboración y atención de pedidos en un local de panificados.

Reseña

La "*Masa madre*" es una forma ancestral de elaboración de pan. Consiste en la cría de un cultivo de bacterias y levaduras naturales presentes en la harina de trigo, sin agregados de levaduras externas. Es un proceso largo que lleva varios días de cuidado y preparación (cada día hay que *alimentar* con harina y agua a la Masa madre)[1]. En las últimas semanas, el estilo de vida hogareño incrementó el interés por este tipo de recetas, que requieren mayor tiempo y cuidado, aunque mucha gente opta por comprar la preparación ya hecha por delivery.

La Masa madre se usa como base, en lugar de la levadura, para elaborar panes, pizza y otros panificados.

Requerimientos Funcionales

La panadería *ConcuBread* elabora panes de campo y pizzas a base de Masa madre y los vende exclusivamente por medio de delivery, recibiendo los pedidos por teléfono.

Los requerimientos funcionales son los siguientes:

1. El maestro especialista en Masa madre la prepara (le agrega periódicamente el "*alimento*").
2. Los maestros panaderos elaboran los panes a partir de la Masa madre, los hornean y los que están listos los van colocando en las grandes canastas.
3. Los maestros pizzeros preparan las pre-pizzas a partir de la Masa madre, las hornean. Cuando les solicitan, preparan las pizzas en el momento a partir de la pre-pizza y el resto de los ingredientes, las cocinan y las colocan en cajas.
4. Los maestros panaderos y pizzeros le solicitan al especialista una ración de Masa madre cuando necesitan.
5. Los recepcionistas del delivery atienden el teléfono y despachan los pedidos de pizza o pan por orden de llegada.
6. Cada pedido es por un pan o una sola pizza y es elaborado por un maestro panadero o pizzero, según corresponda.
7. Las pizzas tardan un tiempo variable en cocinarse.

[1] https://es.wikipedia.org/wiki/Masa_madre

8. Cuando el pedido termina de cocinarse, el cocinero lo pone en una caja y se lo entrega al repartidor que lo lleva al destino.

9. Los siguientes parámetros deben ser configurables sin necesidad de recompilar el código:

   a) Cantidad de recepcionistas del delivery

   b) Cantidad de maestros panaderos y pizzeros

Requerimientos no Funcionales

Los siguientes son los requerimientos no funcionales de la aplicación:

1. El proyecto deberá ser desarrollado en lenguaje C o C++, siendo este último el lenguaje de preferencia.

2. La simulación puede no tener interfaz gráfica y ejecutarse en una o varias consolas de línea de comandos.

3. El proyecto deberá funcionar en ambiente Unix / Linux.

4. La aplicación deberá funcionar en una única computadora.

5. El programa deberá poder ejecutarse en "modo debug", lo cual dejará registro de la actividad que realiza en un único archivo de texto para su revisión posterior. Se deberá poder seguir el recorrido de cada pedido a medida que va pasando por las distintas etapas.

6. Las facilidades de IPC que se podrán utilizar para la realización de este proyecto son las que abarcan la primera parte de la materia, es decir, hasta el primer parcial. Dichas facilidades son:

   a) Memoria compartida

   b) Señales

   c) Pipes y fifos

   d) Locks

   Cualquier otra facilidad queda expresamente excluida para este proyecto.

Tareas a Realizar

A continuación se listan las tareas a realizar para completar el desarrollo del proyecto:

1. Dividir el proyecto en procesos. El objetivo es lograr que la simulación esté conformada por un conjunto de procesos que sean lo más sencillos posible.

2. Una vez obtenida la división en procesos, establecer un esquema de comunicación entre ellos teniendo en cuenta los requerimientos de la aplicación. ¿Qué procesos se comunican entre sí? ¿Qué datos necesitan compartir para poder trabajar?

3. Tratar de mapear la comunicación entre los procesos a los problemas conocidos de concurrencia.

4. Determinar los mecanismos de concurrencia a utilizar para cada una de las comunicaciones entre procesos que fueron detectadas en el ítem 2. No se requiere la utilización de algún mecanismo específico, la elección en cada caso queda a cargo del grupo y debe estar debidamente justificada.

5. Realizar la codificación de la aplicación. El código fuente debe estar documentado.

Entrega

La entrega del proyecto comprende lo siguiente:

1. Informe, se deberá presentar impreso en una carpeta o folio y en forma digital (PDF) a través del campus
2. El código fuente de la aplicación, que se entregará únicamente mediante el campus

La entrega en el campus estará habilitada hasta las 19 hs de la fecha indicada oportunamente.

El informe a entregar debe contener los siguientes ítems:

1. Breve análisis del problema, incluyendo una especificación de los casos de uso de la aplicación.
2. Detalle de resolución de la lista de tareas anterior.
3. Diagrama que refleje los procesos, el flujo de comunicación entre ellos y los datos que intercambian.
4. Diagramas de clases realizados.
5. Diagrama de transición de estados de un pedido.

```cpp
//
// Created by rozanecm on 5/12/20.
//

#include "Actor.h"

#include <utility>
#include <iostream>

Actor::Actor() {
    canal_debug.abrir();
}

Actor::~Actor() {
    canal_debug.cerrar();
}

void Actor::mandar_msj_fifo(std::string mensaje, FifoEscritura *fifo) {
    fifo->escribir(static_cast<const void *>(mensaje.c_str()),
                   mensaje.length());
}

void Actor::mandar_msj_debug(std::string mensaje) {
    mensaje += "\n";
    mandar_msj_fifo(std::move(mensaje), &canal_debug);
}
```

```
1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #ifndef CONCUBREAD_ACTOR_H
6  #define CONCUBREAD_ACTOR_H
7
8
9  #include "../../fifos/FifoEscritura.h"
10 #include "../../utils/consts.h"
11 #include "../Proceso.h"
12
13 class Actor : public Proceso {
14 public:
15     Actor();
16
17     ~Actor();
18
19     void mandar_msj_fifo(std::string mensaje, FifoEscritura *fifo);
20
21     void mandar_msj_debug(std::string mensaje);
22
23 protected:
24     FifoEscritura canal_debug = FifoEscritura(ARCHIVO_FIFO_DEBUG);
25 };
26
27
28 #endif //CONCUBREAD_ACTOR_H
```

```
1   //
2   // Created by rozanecm on 5/12/20.
3   //
4
5   #include <iostream>
6   #include <random>
7   #include "Cocinero.h"
8
9   Cocinero::Cocinero() {
10      this->fl.l_type = F_RDLCK;
11      this->fl.l_whence = SEEK_SET;
12      this->fl.l_start = 0;
13      this->fl.l_len = 0;
14
15      canal_envio_pedidos_especialista_MM.abrir();
16  }
17
18  Cocinero::~Cocinero() {
19      canal_envio_pedidos_especialista_MM.cerrar();
20      canal_recepcion_de_mm->cerrar();
21      // eliminacion en void EspecialistaMasaMadre::cerrar_canales_particulares();
22  }
23
24  void Cocinero::ejercer_tarea() {
25      bool seguimos_recibiendo = true;
26      while (sigint_handler.getGracefulQuit() == 0 and seguimos_recibiendo) {
27          recibir_pedido(&seguimos_recibiendo);
28          if (seguimos_recibiendo) {
29              realizar_pedido();
30              entregar_pedido_a_repartidor();
31          }
32      }
33      mandar_msj_debug(id + " termino con todos sus deberes.");
34  }
35
36  void Cocinero::recibir_pedido(bool *seguir_recibiendo_pedidos) {
37      ssize_t bytesLeidos = 0;
38      this->id_pedido_actual = "";
39
40      fcntl(canal_recepcionista->get_fd(), F_SETLKW, &(this->fl));
41
42      bytesLeidos += canal_recepcionista->leer(static_cast<void *>
43                                               (buffer_recepcionista),
44                                               FIFO_PEDIDOS_BUFFSIZE);
45
46      fcntl(canal_recepcionista->get_fd(), F_SETLK, &(this->fl));
47
48      if (bytesLeidos == 0) {
49          *seguir_recibiendo_pedidos = false;
50      } else {
51          this->id_pedido_actual = buffer_recepcionista;
52          mandar_msj_debug(id + ". Recibi pedido: " +
53                           this->id_pedido_actual);
54      }
55  }
56
57  void Cocinero::realizar_pedido() {
58      /* Inicializacion random */
59      std::random_device rd;
60      std::mt19937 gen(rd());
61      std::uniform_int_distribution<> dis(1, 6);
62
63      pedir_racion_mm();
64      esperar_envio_mm();
65      mandar_msj_debug(id + " realizando pedido " + id_pedido_actual);
66      sleep(dis(gen));
67      mandar_msj_debug(id + " termine de realizar el pedido " +
68                       id_pedido_actual);
69  }
70
71  void Cocinero::pedir_racion_mm() {
72      mandar_msj_debug(id + " pidiendo racion de masa madre...");
73      mandar_msj_fifo(id, &this->canal_envio_pedidos_especialista_MM);
```



```
74  }
75
76  void Cocinero::esperar_envio_mm() {
77      abrir_canal_recepcion_de_mm();
78      std::string mensaje;
79      canal_recepcion_de_mm->leer(static_cast<void *>(buffer_recepcion_mm),
80                                  LENGTH_MSJ_ENVIO_MM);
81      mensaje = buffer_recepcion_mm;
82      mensaje.resize(LENGTH_MSJ_ENVIO_MM);
83      if (mensaje == MENSAJE_PARA_ENVIAR_MM_A_COCINEROS) {
84          mandar_msj_debug(id + " recibi racion de masa madre! "
85                                "Continuemos con el pedido...");
86      } else {
87          mandar_msj_debug("ALERTA! " + id + " Por alguna razon no recibi"
88                                                 " racion masa madre y puedo seguir igual...");
89          std::cout << "msj recibido en lugar de mm: " << mensaje << std::endl;
90          mandar_msj_debug("msj recibido en lu   gar de mm: " +
91                           std::to_string(int(buffer_recepcion_mm[1])));
92      }
93  }
94
95  void Cocinero::entregar_pedido_a_repartidor() {
96      mandar_msj_debug(id + " entregando pedido " + id_pedido_actual +
97                       " a repartidor.");
98  }
99
100 void Cocinero::abrir_canal_recepcion_de_mm() {
101     if (not canal_recepcion_de_mm_fue_abierto) {
102         canal_recepcion_de_mm = std::make_unique<FifoLectura>(id);
103         canal_recepcion_de_mm->abrir();
104         canal_recepcion_de_mm_fue_abierto = true;
105     }
106 }
```

```
1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #ifndef CONCUBREAD_COCINERO_H
6  #define CONCUBREAD_COCINERO_H
7
8  #include <string>
9  #include <memory>
10 #include "Actor.h"
11 #include "../../fifos/FifoLectura.h"
12
13 class Cocinero : public Actor {
14 public:
15     explicit Cocinero();
16
17     ~Cocinero();
18
19     void ejercer_tarea();
20
21     void recibir_pedido(bool *seguir_recibiendo_pedidos);
22
23     void realizar_pedido();
24
25     void entregar_pedido_a_repartidor();
26
27     void pedir_racion_mm();
28
29 protected:
30     std::string id;
31     struct flock fl;
32     std::string id_pedido_actual;
33
34     std::unique_ptr<FifoLectura> canal_recepcionista;
35     char buffer_recepcionista[FIFO_PEDIDOS_BUFFSIZE];
36
37 private:
38     FifoEscritura canal_envio_pedidos_especialista_MM =
39             FifoEscritura(ARCHIVO_FIFO_PEDIDOS_MM);
40
41     std::unique_ptr<FifoLectura> canal_recepcion_de_mm;
42     char buffer_recepcion_mm[LENGTH_MSJ_ENVIO_MM];
43
44     void esperar_envio_mm();
45
46     void abrir_canal_recepcion_de_mm();
47
48     bool canal_recepcion_de_mm_fue_abierto = false;
49 };
50
51 #endif //CONCUBREAD_COCINERO_H
```

```
1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #ifndef CONCUBREAD_CONSTS_H
6  #define CONCUBREAD_CONSTS_H
7
8  #define FIFO_DEBUG_BUFFSIZE 100
9
10 /* el id que se manda con el pedido tiene siempre esta longitud */
11 #define FIFO_PEDIDOS_BUFFSIZE 7
12 /* el msj que le mandan es el id, que tiene 5 chars siempre
13  * (Piz/Pan + id de dos digitos) */
14 #define FIFO_ESPECIALISTA_MM_BUFFSIZE 5
15
16 /* Se definen las siguientes ctes. para entender mas facilmente los
17  * valores en el codigo */
18 #define MENSAJE_PARA_ENVIAR_MM_A_COCINEROS "X"
19 #define LENGTH_MSJ_ENVIO_MM 1
20
21 #define ARCHIVO_FIFO_DEBUG "/tmp/archivo_debug"
22 #define ARCHIVO_FIFO_PANADEROS "/tmp/archivo_fifo_panaderos"
23 #define ARCHIVO_FIFO_PIZZEROS "/tmp/archivo_fifo_pizzeros"
24 #define ARCHIVO_FIFO_PEDIDOS_MM "/tmp/archivo_fifo_pedidos_MM"
25
26
27 #endif //CONCUBREAD_CONSTS_H
```

```cpp
//
// Created by rozanecm on 5/12/20.
//

#include <iostream>
#include "DebugPrinter.h"
#include "../signals/SignalHandler.h"

#define DEBUG_FILE_PATH "../log_info.log"

DebugPrinter::DebugPrinter(bool should_i_print) : should_i_print(
        should_i_print) {
    canal_debug.abrir();
    debug_file.open(DEBUG_FILE_PATH, std::ios::out | std::ios::trunc);
    SignalHandler::getInstance()->registrarHandler(SIGINT, &sigint_handler);
}

DebugPrinter::~DebugPrinter() {
    canal_debug.cerrar();
    canal_debug.eliminar();
    debug_file.close();
    SignalHandler::destruir();
}

void DebugPrinter::atender_debug_msgs() {
    if (should_i_print) {
        ssize_t bytesLeidos = 1;

        bool seguir_escuchando = true;
        while (sigint_handler.getGracefulQuit() == 0 and seguir_escuchando) {
            bytesLeidos = canal_debug.leer(static_cast<void *>(buffer),
                                           FIFO_DEBUG_BUFFSIZE);
            if (bytesLeidos > 0) {
                std::string mensaje = buffer;
                mensaje.resize(bytesLeidos);
                this->print(mensaje);
            } else {
                seguir_escuchando = false;
            }
        }
    }
}

void DebugPrinter::ejercer_tarea() {
    atender_debug_msgs();
}

void DebugPrinter::print(const std::string &msg) {
    debug_file << msg << std::flush;
}
```

```
1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #ifndef CONCUBREAD_DEBUGPRINTER_H
6  #define CONCUBREAD_DEBUGPRINTER_H
7
8
9  #include <fstream>
10 #include "../signals/SIGINT_Handler.h"
11 #include "../fifos/FifoLectura.h"
12 #include "../utils/consts.h"
13 #include "Proceso.h"
14
15 class DebugPrinter : public Proceso {
16 public:
17     DebugPrinter(bool should_i_print);
18
19     ~DebugPrinter();
20
21     void ejercer_tarea() override;
22
23 private:
24     bool should_i_print;
25     FifoLectura canal_debug = FifoLectura(ARCHIVO_FIFO_DEBUG);
26     char buffer[FIFO_DEBUG_BUFFSIZE];
27
28     void atender_debug_msgs();
29
30     void print(const std::string &);
31
32     std::ofstream debug_file;
33 };
34
35
36 #endif //CONCUBREAD_DEBUGPRINTER_H
```

```cpp
//
// Created by rozanecm on 5/12/20.
//

#include <iostream>
#include "EspecialistaMasaMadre.h"


EspecialistaMasaMadre::EspecialistaMasaMadre() {
    canal_cocineros.abrir();
}

EspecialistaMasaMadre::~EspecialistaMasaMadre() {
    canal_cocineros.cerrar();
    canal_cocineros.eliminar();
    cerrar_canales_particulares();
}

void EspecialistaMasaMadre::ejercer_tarea() {
    bool seguir_recibiendo_pedidos = true;
    while (sigint_handler.getGracefulQuit() == 0 and seguir_recibiendo_pedidos)
    {
        std::string id_solicitante = escuchar_pedidos(
                &seguir_recibiendo_pedidos);
        if (seguir_recibiendo_pedidos) {
            mandar_msj_debug("Especialista: recibi pedido de racion de"
                             " MM de parte de: " + id_solicitante);
            entregar_pedido(id_solicitante);
        }
    }
}

std::string EspecialistaMasaMadre::escuchar_pedidos(bool *
seguir_recibiendo_pedidos) {
    ssize_t bytesLeidos = 0;
    std::string id_solicitante_actual;

    while (bytesLeidos < FIFO_ESPECIALISTA_MM_BUFFSIZE) {
        bytesLeidos += canal_cocineros.leer(static_cast<void *>(buffer),
                                            FIFO_ESPECIALISTA_MM_BUFFSIZE);
        std::string mensaje = buffer;
        mensaje.resize(bytesLeidos);
        if (bytesLeidos == 0) {
            *seguir_recibiendo_pedidos = false;
            break;
        }
        id_solicitante_actual.append(mensaje);
    }
    return id_solicitante_actual;
}

void EspecialistaMasaMadre::entregar_pedido(const std::string &id_solicitante) {
    obtener_canal_envio_mm(id_solicitante)->escribir(
            MENSAJE_PARA_ENVIAR_MM_A_COCINEROS, LENGTH_MSJ_ENVIO_MM);
}

bool
EspecialistaMasaMadre::existe_canal_envio_mm(const std::string &id_pedido) {
    for (auto &it : canales_envio_mm) {
        if (it.first == id_pedido) {
            return true;
        }
    }
    return false;
}

FifoEscritura *EspecialistaMasaMadre::obtener_canal_envio_mm(
        const std::string &id_solicitante) {
    if (not existe_canal_envio_mm(id_solicitante)) {
        canales_envio_mm.push_back(std::make_pair(id_solicitante,
                                                  new FifoEscritura(
                                                          id_solicitante)));
        canales_envio_mm.back().second->abrir();
```



```cpp
        return canales_envio_mm.back().second;
    }
    for (auto &it : canales_envio_mm) {
        if (it.first == id_solicitante)
            return it.second;
    }
}

void EspecialistaMasaMadre::cerrar_canales_particulares() {
    for (auto &it : canales_envio_mm) {
        it.second->cerrar();
        it.second->eliminar();
        delete it.second;
    }
    canales_envio_mm.clear();
}
```

```
1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #ifndef CONCUBREAD_ESPECIALISTAMASAMADRE_H
6  #define CONCUBREAD_ESPECIALISTAMASAMADRE_H
7
8
9  #include <unordered_map>
10 #include <map>
11 #include <vector>
12 #include <memory>
13 #include "Actor.h"
14 #include "../../fifos/FifoLectura.h"
15
16 class EspecialistaMasaMadre : public Actor {
17 public:
18     EspecialistaMasaMadre();
19
20     ~EspecialistaMasaMadre();
21
22     void ejercer_tarea() override;
23
24 private:
25     FifoLectura canal_cocineros = FifoLectura(ARCHIVO_FIFO_PEDIDOS_MM);
26     char buffer[FIFO_ESPECIALISTA_MM_BUFFSIZE];
27
28     std::string escuchar_pedidos(bool *seguir_recibiendo_pedidos);
29
30     void entregar_pedido(const std::string &id_solicitante);
31
32     bool existe_canal_envio_mm(const std::string &id_pedido);
33
34     void cerrar_canales_particulares();
35
36     FifoEscritura *obtener_canal_envio_mm(const std::string &id_solicitante);
37
38     std::vector<std::pair<std::string, FifoEscritura *>> canales_envio_mm;
39 };
40
41
42 #endif //CONCUBREAD_ESPECIALISTAMASAMADRE_H
```

```
1  #ifndef EVENTHANDLER_H_
2  #define EVENTHANDLER_H_
3
4  class EventHandler {
5
6  public:
7      virtual int handleSignal(int signum) = 0;
8
9      virtual ~EventHandler() {};
10 };
11
12 #endif /* EVENTHANDLER_H_ */
```

```cpp
#include "Fifo.h"

Fifo::Fifo(const std::string nombre) : nombre(nombre), fd(-1) {
    mknod(static_cast<const char *>(nombre.c_str()), S_IFIFO | 0666, 0);
}

Fifo::~Fifo() {
}

void Fifo::cerrar() {
    close(fd);
    fd = -1;
}

void Fifo::eliminar() const {
    unlink(nombre.c_str());
}

int Fifo::get_fd() {
    return fd;
}
```

```
1  #include "FifoEscritura.h"
2
3  FifoEscritura::FifoEscritura(const std::string nombre) : Fifo(nombre) {
4  }
5
6  FifoEscritura::~FifoEscritura() {
7  }
8
9  void FifoEscritura::abrir() {
10     fd = open(nombre.c_str(), O_WRONLY);
11 }
12
13 ssize_t
14 FifoEscritura::escribir(const void *buffer, const ssize_t buffsize) const {
15     return write(fd, buffer, buffsize);
16 }
```

```
1  #ifndef FIFOESCRITURA_H_
2  #define FIFOESCRITURA_H_
3
4  #include "Fifo.h"
5
6  class FifoEscritura : public Fifo {
7  public:
8      FifoEscritura(const std::string nombre);
9
10     ~FifoEscritura();
11
12     void abrir();
13
14     ssize_t escribir(const void *buffer, const ssize_t buffsize) const;
15 };
16
17 #endif /* FIFOESCRITURA_H_ */
```

```
1  #ifndef FIFO_H_
2  #define FIFO_H_
3
4  #include <string>
5  #include <sys/types.h>
6  #include <sys/stat.h>
7  #include <fcntl.h>
8  #include <unistd.h>
9
10 class Fifo {
11 public:
12         Fifo(const std::string nombre);
13         virtual ~Fifo();
14         virtual void abrir() = 0;
15         void cerrar();
16         void eliminar() const;
17
18     int get_fd();
19
20 protected:
21         std::string nombre;
22         int fd;
23 };
24
25 #endif /* FIFO_H_ */
```

```cpp
#include "FifoLectura.h"

FifoLectura::FifoLectura(const std::string nombre) : Fifo(nombre) {
}

FifoLectura::~FifoLectura() {
}

void FifoLectura::abrir() {
    fd = open(nombre.c_str(), O_RDONLY);
}

ssize_t FifoLectura::leer(void *buffer, const ssize_t buffsize) const {
    return read(fd, buffer, buffsize);
}
```

```
1  #ifndef FIFOLECTURA_H_
2  #define FIFOLECTURA_H_
3
4  #include "Fifo.h"
5
6  class FifoLectura : public Fifo {
7  public:
8      FifoLectura(const std::string nombre);
9
10     ~FifoLectura();
11
12     void abrir();
13
14     ssize_t leer(void *buffer, const ssize_t buffsize) const;
15 };
16
17 #endif /* FIFOLECTURA_H_ */
```

```
#include <iostream>
#include <fstream>
#include <cstring>
#include "utils/ProcessManager.h"

#define ARCHIVO_CONFIG_PATH "../config.cb"

void leer_config_file(int *cant_panaderos, int *cant_pizzeros,
                      int *cant_recepcionistas);

bool print_debug_msgs(int argc, char **argv);

int main(int argc, char *argv[]) {
    int cant_panaderos, cant_pizzeros, cant_recepcionistas;
    leer_config_file(&cant_panaderos, &cant_pizzeros, &cant_recepcionistas);

    std::cout << "Bienvenido a ConcuBread!" << std::endl;
    std::cout << "Simulando..." << std::endl;

    const std::unique_ptr<Proceso> &proceso_generado =
            ProcessManager::crear_procesos(
                    cant_panaderos, cant_pizzeros,
                    cant_recepcionistas,
                    print_debug_msgs(argc, argv));
    proceso_generado->ejercer_tarea();
    return 0;
}

bool print_debug_msgs(int argc, char **argv) {
    if (argc > 1) {
        if (std::strcmp(argv[1], "--print-debug") == 0) {
            if (std::strcmp(argv[2], "false") == 0) {
                std::cout << "Setting debug printing off..." << std::endl;
                return false;
            }
            if (std::strcmp(argv[2], "true") == 0) {
                std::cout << "Setting debug printing on..." << std::endl;
                return true;
            }
        }
        std::cout << "Unidentified arg name. Debug printing set on as default."
                  << std::endl;
        return true;
    }
    /* si no se especifica nada... */
    return true;
}

void leer_config_file(int *cant_panaderos, int *cant_pizzeros,
                      int *cant_recepcionistas) {
    std::ifstream config_file(ARCHIVO_CONFIG_PATH);
    if (config_file.is_open()) {
        std::string line;
        try {
            std::getline(config_file, line);
            *cant_panaderos = std::stoi(line);

            std::getline(config_file, line);
            *cant_pizzeros = std::stoi(line);

            std::getline(config_file, line);
            *cant_recepcionistas = std::stoi(line);
        } catch (std::exception const &ex) {
            std::cout
                    << "Error reading config file."
                       " Please check config file format and try again."
                    << std::endl;
            config_file.close();
            exit(0);
        }
        config_file.close();
    } else {
        std::cout
```



```
                << "No config file found!"
                   " Please, verify the config file is present and try again."
                << std::endl;
        exit(0);
    }
}
```

```
1  //
2  // Created by rozanecm on 5/21/20.
3  //
4
5  #include <sys/wait.h>
6  #include <iostream>
7  #include "Padre.h"
8
9  void Padre::ejercer_tarea() {
10     reportarse();
11
12     bool volvieron_todos = false;
13     while (sigint_handler.getGracefulQuit() == 0 and not volvieron_todos) {
14         int status = 0;
15         auto wait_val = wait(&status);
16         if (wait_val < 0) {
17             volvieron_todos = true;
18         }
19     }
20 }
21
22 void Padre::reportarse() {
23     canal_debug.abrir();
24     std::string mensaje =
25             "Soy el padre con pid " + std::to_string(getpid()) + '\n';
26     canal_debug.escribir(static_cast<const void *>(mensaje.c_str()),
27                          mensaje.length());
28     canal_debug.cerrar();
29 }
```

```
1  //
2  // Created by rozanecm on 5/21/20.
3  //
4
5  #ifndef CONCUBREAD_PADRE_H
6  #define CONCUBREAD_PADRE_H
7
8
9  #include "Actores/Actor.h"
10
11 class Padre : public Proceso {
12     void ejercer_tarea() override;
13
14 private:
15     FifoEscritura canal_debug = FifoEscritura(ARCHIVO_FIFO_DEBUG);
16
17     void reportarse();
18 };
19
20
21 #endif //CONCUBREAD_PADRE_H
```

```cpp
//
// Created by rozanecm on 5/12/20.
//

#include <iostream>
#include "Panadero.h"

Panadero::Panadero(int id_panadero) {
    this->canal_recepcionista = std::make_unique<FifoLectura>(
            ARCHIVO_FIFO_PANADEROS);
    canal_recepcionista->abrir();

    set_id(id_panadero);
}

void Panadero::set_id(int id_panadero) {
    id = "Pan";
    if (id_panadero < 10) {
        id.append("0");
    }
    id.append(std::to_string(id_panadero));
}

Panadero::~Panadero() {
    canal_recepcionista->cerrar();
}
```

```
1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #ifndef CONCUBREAD_PANADERO_H
6  #define CONCUBREAD_PANADERO_H
7
8
9  #include "Cocinero.h"
10 #include "../../fifos/FifoLectura.h"
11
12 class Panadero : public Cocinero {
13 public:
14     explicit Panadero(int id_panadero);
15
16     ~Panadero();
17
18     void set_id(int id_panadero);
19 };
20
21
22 #endif //CONCUBREAD_PANADERO_H
```

```cpp
//
// Created by rozanecm on 5/12/20.
//

#include <iostream>
#include "Pizzero.h"

Pizzero::Pizzero(int id_pizzero) {
    this->canal_recepcionista = std::make_unique<FifoLectura>
            (ARCHIVO_FIFO_PIZZEROS);
    canal_recepcionista->abrir();

    set_id(id_pizzero);
}

Pizzero::~Pizzero() {
    canal_recepcionista->cerrar();
}

void Pizzero::set_id(int id_pizzero) {
    id = "Piz";
    if (id_pizzero < 10) {
        id.append("0");
    }
    id.append(std::to_string(id_pizzero));
}
```

```
1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #ifndef CONCUBREAD_PIZZERO_H
6  #define CONCUBREAD_PIZZERO_H
7
8
9  #include "Cocinero.h"
10 #include "../../fifos/FifoLectura.h"
11
12 class Pizzero : public Cocinero {
13 public:
14     explicit Pizzero(int id_pizzero);
15
16     ~Pizzero();
17
18     void set_id(int id_pizzero);
19 };
20
21
22 #endif //CONCUBREAD_PIZZERO_H
```

```cpp
//
// Created by rozanecm on 5/13/20.
//

#include "Proceso.h"
```

```
1  //
2  // Created by rozanecm on 5/13/20.
3  //
4
5  #ifndef CONCUBREAD_PROCESO_H
6  #define CONCUBREAD_PROCESO_H
7
8
9  #include "../signals/SIGINT_Handler.h"
10
11 class Proceso {
12 public:
13     virtual ~Proceso() {};
14
15     virtual void ejercer_tarea() = 0;
16
17 protected:
18     SIGINT_Handler sigint_handler;
19 };
20
21
22 #endif //CONCUBREAD_PROCESO_H
```

```cpp
//
// Created by rozanecm on 5/6/20.
//

#include "ProcessManager.h"
#include "../Procesos/DebugPrinter.h"
#include "../Procesos/Actores/Panadero.h"
#include "../Procesos/Actores/Pizzero.h"
#include "../Procesos/Actores/Recepcionista.h"
#include "../Procesos/Actores/EspecialistaMasaMadre.h"
#include "../Procesos/Padre.h"
#include <unistd.h>
#include <bits/unique_ptr.h>

std::unique_ptr<Proceso>
ProcessManager::crear_procesos(int cant_panaderos, int cant_pizzeros,
                               int cant_recepcionistas, bool print_debug_msgs) {
    /* Creacion panaderos */
    for (int i = 0; i < cant_panaderos; ++i) {
        if (fork() == 0) {
            return std::make_unique<Panadero>(i);
        }
    }
    /* Creacion pizzeros */
    for (int i = 0; i < cant_pizzeros; ++i) {
        if (fork() == 0) {
            return std::make_unique<Pizzero>(i);
        }
    }
    /* Creacion recepcionistas */
    for (int i = 0; i < cant_recepcionistas; ++i) {
        if (fork() == 0) {
            return std::make_unique<Recepcionista>(i);
        }
    }
    /* Creacion espec. masa madre */
    if (fork() == 0) {
        return std::make_unique<EspecialistaMasaMadre>();
    }
    /* Creacion Debug Printer */
    if (fork() == 0) {
        return std::make_unique<DebugPrinter>(print_debug_msgs);
    }
    return std::make_unique<Padre>();
}
```

```
1  //
2  // Created by rozanecm on 5/6/20.
3  //
4
5  #ifndef CONCUBREAD_PROCESSMANAGER_H
6  #define CONCUBREAD_PROCESSMANAGER_H
7
8
9  #include <bits/unique_ptr.h>
10 #include "../Procesos/Proceso.h"
11
12 class ProcessManager {
13 public:
14     static std::unique_ptr<Proceso>
15     crear_procesos(int cant_panaderos, int cant_pizzeros,
16                    int cant_recepcionistas, bool print_debug_msgs);
17 };
18
19
20 #endif //CONCUBREAD_PROCESSMANAGER_H
```

```cpp
//
// Created by rozanecm on 5/12/20.
//

#include <iostream>
#include <random>
#include "Recepcionista.h"

#define CANT_PIZZAS 2
#define CANT_PANES 1

Recepcionista::Recepcionista(int id) : id(id) {
    canal_pizzeros.abrir();
    canal_panaderos.abrir();
}

Recepcionista::~Recepcionista() {
    canal_pizzeros.cerrar();
    canal_panaderos.cerrar();
    canal_pizzeros.eliminar();
    canal_panaderos.eliminar();
}

void Recepcionista::ejercer_tarea() {
    hacer_pedidos();
}

void Recepcionista::hacer_pedidos() {
    /* La parte random esta sacada de
     * https://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution
     * */
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(1, 6);

    auto cant_pizzas_a_encargar = dis(gen);
    auto cant_panes_a_encargar = dis(gen);

    auto decision_pizza_o_pan = dis(gen);

    int pizzas_encargadas = 0;
    int panes_encargados = 0;
    while (sigint_handler.getGracefulQuit() == 0 and
           (pizzas_encargadas < cant_pizzas_a_encargar or
            panes_encargados < cant_panes_a_encargar)) {
        if (decision_pizza_o_pan % 2 == 0 and
            pizzas_encargadas < cant_pizzas_a_encargar) {
            pizzas_encargadas++;
            pedir_pizza(pizzas_encargadas, cant_pizzas_a_encargar);
            sleep(dis(gen));
        } else if (panes_encargados < cant_panes_a_encargar) {
            panes_encargados++;
            pedir_pan(panes_encargados, cant_panes_a_encargar);
            sleep(dis(gen));
        }
        decision_pizza_o_pan = dis(gen);
    }
}

void Recepcionista::pedir_pizza(int numero_de_encargo_pizza,
                                int cant_total_de_pizzas_a_hacer) {
    hacer_pedido_pizzeros(numero_de_encargo_pizza,
                          cant_total_de_pizzas_a_hacer);
}

void Recepcionista::pedir_pan(int numero_de_encargo_pan,
                              int cant_total_de_panes_a_hacer) {
    hacer_pedido_panaderos(numero_de_encargo_pan, cant_total_de_panes_a_hacer);
}

void Recepcionista::hacer_pedido_pizzeros(int numero_de_encargo_pizza,
                                          int cant_total_de_pizzas_a_hacer) {
    auto id_pizza = id_prox_pizza();
```



```cpp

    mandar_msj_debug("Recepc. " + std::to_string(id) +
                     " Encargando pizza " + id_pizza + " (" +
                     std::to_string(numero_de_encargo_pizza) + "/" +
                     std::to_string(cant_total_de_pizzas_a_hacer) + ")");

    mandar_msj_fifo(id_pizza, &canal_pizzeros);
    cantidad_pizzas_encargadas++;
}

void Recepcionista::hacer_pedido_panaderos(int numero_de_encargo_pan,
                                           int cant_total_de_panes_a_hacer) {
    auto id_pan = id_prox_pan();

    mandar_msj_debug("Recepc. " + std::to_string(id) +
                     " Encargando pan " + id_pan + " (" +
                     std::to_string(numero_de_encargo_pan) + "/" +
                     std::to_string(cant_total_de_panes_a_hacer) + ")");

    mandar_msj_fifo(id_pan, &canal_panaderos);
    cantidad_panes_encargados++;
}

std::string Recepcionista::id_prox_pizza() const {
    std::string id_pedido = "Piz";
    if (this->id < 10) {
        id_pedido += "0";
    }
    id_pedido += std::to_string(this->id);
    if (this->cantidad_pizzas_encargadas < 10) {
        id_pedido += "0";
    }
    id_pedido += std::to_string(cantidad_pizzas_encargadas);
    return id_pedido;
}

std::string Recepcionista::id_prox_pan() const {
    std::string id_pedido = "Pan";
    if (this->id < 10) {
        id_pedido += "0";
    }
    id_pedido += std::to_string(this->id);
    if (this->cantidad_panes_encargados < 10) {
        id_pedido += "0";
    }
    id_pedido += std::to_string(cantidad_panes_encargados);
    return id_pedido;
}
```

```
1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #ifndef CONCUBREAD_RECEPCIONISTA_H
6  #define CONCUBREAD_RECEPCIONISTA_H
7
8  #include "Actor.h"
9
10 class Recepcionista : public Actor {
11 public:
12     explicit Recepcionista(int id);
13
14     ~Recepcionista();
15
16     void ejercer_tarea() override;
17
18 private:
19     void pedir_pan(int numero_de_encargo_pan, int cant_total_de_panes_a_hacer);
20
21     void pedir_pizza(int numero_de_encargo_pizza,
22                      int cant_total_de_pizzas_a_hacer);
23
24     int id;
25
26     void hacer_pedidos();
27
28     int cantidad_pizzas_encargadas = 0;
29     int cantidad_panes_encargados = 0;
30
31     FifoEscritura canal_panaderos = FifoEscritura(ARCHIVO_FIFO_PANADEROS);
32     FifoEscritura canal_pizzeros = FifoEscritura(ARCHIVO_FIFO_PIZZEROS);
33
34     void hacer_pedido_panaderos(int numero_de_encargo_pan,
35                                 int cant_total_de_panes_a_hacer);
36
37     void hacer_pedido_pizzeros(int numero_de_encargo_pizza,
38                                int cant_total_de_pizzas_a_hacer);
39
40     std::string id_prox_pizza() const;
41
42     std::string id_prox_pan() const;
43 };
44
45
46 #endif //CONCUBREAD_RECEPCIONISTA_H
```

```
1  #ifndef SIGINT_HANDLER_H_
2  #define SIGINT_HANDLER_H_
3
4  #include <signal.h>
5  #include <assert.h>
6
7  #include "EventHandler.h"
8
9  class SIGINT_Handler : public EventHandler {
10
11 private:
12     sig_atomic_t gracefulQuit;
13
14 public:
15
16     SIGINT_Handler() : gracefulQuit(0) {
17     }
18
19     ~SIGINT_Handler() {
20     }
21
22     virtual int handleSignal(int signum) {
23         assert (signum == SIGINT);
24         this->gracefulQuit = 1;
25         return 0;
26     }
27
28     sig_atomic_t getGracefulQuit() const {
29         return this->gracefulQuit;
30     }
31
32 };
33
34 #endif /* SIGINT_HANDLER_H_ */
```

```
1  #include "SignalHandler.h"
2
3  SignalHandler *SignalHandler::instance = NULL;
4  EventHandler *SignalHandler::signal_handlers[NSIG];
5
6  SignalHandler::SignalHandler() {
7  }
8
9  SignalHandler *SignalHandler::getInstance() {
10
11     if (instance == NULL)
12         instance = new SignalHandler();
13
14     return instance;
15 }
16
17 void SignalHandler::destruir() {
18     if (instance != NULL) {
19         delete (instance);
20         instance = NULL;
21     }
22 }
23
24 EventHandler *SignalHandler::registrarHandler(int signum, EventHandler *eh) {
25
26     EventHandler *old_eh = SignalHandler::signal_handlers[signum];
27     SignalHandler::signal_handlers[signum] = eh;
28
29     struct sigaction sa;
30     memset(&sa, 0, sizeof(sa));
31     sa.sa_handler = SignalHandler::dispatcher;
32     sigemptyset(
33             &sa.sa_mask);    // inicializa la mascara de seniales a bloquear
34     // durante la ejecucion del handler como vacio
35     sigaddset(&sa.sa_mask, signum);
36     sigaction(signum, &sa, 0);    // cambiar accion de la senial
37
38     return old_eh;
39 }
40
41 void SignalHandler::dispatcher(int signum) {
42
43     if (SignalHandler::signal_handlers[signum] != 0)
44         SignalHandler::signal_handlers[signum]->handleSignal(signum);
45 }
46
47 int SignalHandler::removerHandler(int signum) {
48
49     SignalHandler::signal_handlers[signum] = NULL;
50     return 0;
51 }
```

```
1  #ifndef SIGNALHANDLER_H_
2  #define SIGNALHANDLER_H_
3
4  #include <signal.h>
5  #include <stdio.h>
6  #include <memory.h>
7
8  #include "EventHandler.h"
9
10 class SignalHandler {
11
12 private:
13     static SignalHandler *instance;
14     static EventHandler *signal_handlers[NSIG];
15
16     SignalHandler(void);
17
18     static void dispatcher(int signum);
19
20 public:
21     static SignalHandler *getInstance();
22
23     static void destruir();
24
25     EventHandler *registrarHandler(int signum, EventHandler *eh);
26
27     int removerHandler(int signum);
28
29 };
30
31 #endif /* SIGNALHANDLER_H_ */
```