

May 24, 20 1:11

Actor.cpp

Page 1/1

```
1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #include "Actor.h"
6
7  #include <utility>
8  #include <iostream>
9
10 Actor::Actor() {
11     canal_debug.abrir();
12 }
13
14 Actor::~Actor() {
15     canal_debug.cerrar();
16 }
17
18 void Actor::mandar_msj_fifo(std::string mensaje, FifoEscritura *fifo) {
19     fifo->escribir(static_cast<const void *>(mensaje.c_str()),
20                   mensaje.length());
21 }
22
23 void Actor::mandar_msj_debug(std::string mensaje) {
24     mensaje += "\n";
25     mandar_msj_fifo(std::move(mensaje), &canal_debug);
26 }
```

May 24, 20 1:11

Actor.h

Page 1/1

```
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
```

May 24, 20 1:20

Cocinero.cpp

Page 1/2

```

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_SETLK, &(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);

```

May 24, 20 1:20

Cocinero.cpp

Page 2/2

```

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 }

```

May 24, 20 1:12

Cocinero.h

Page 1/1

```

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

```

May 24, 20 1:15

consts.h

Page 1/1

```
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
```

May 24, 20 1:13

DebugPrinter.cpp

Page 1/1

```

1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #include <iostream>
6  #include "DebugPrinter.h"
7  #include "../signals/SignalHandler.h"
8
9  #define DEBUG_FILE_PATH "../log_info.log"
10
11 DebugPrinter::DebugPrinter(bool should_i_print) : should_i_print(
12     should_i_print) {
13     canal_debug.abrir();
14     debug_file.open(DEBUG_FILE_PATH, std::ios::out | std::ios::trunc);
15     SignalHandler::getInstance()->registrarHandler(SIGINT, &sigint_handler);
16 }
17
18 DebugPrinter::~DebugPrinter() {
19     canal_debug.cerrar();
20     canal_debug.eliminar();
21     debug_file.close();
22     SignalHandler::destruir();
23 }
24
25 void DebugPrinter::atender_debug_msgs() {
26     if (should_i_print) {
27         ssize_t bytesLeidos = 1;
28
29         bool seguir_escuchando = true;
30         while (sigint_handler.getGracefulQuit() == 0 and seguir_escuchando) {
31             bytesLeidos = canal_debug.leer(static_cast<void *>(buffer),
32                                           FIFO_DEBUG_BUFFSIZE);
33
34             if (bytesLeidos > 0) {
35                 std::string mensaje = buffer;
36                 mensaje.resize(bytesLeidos);
37                 this->print(mensaje);
38             } else {
39                 seguir_escuchando = false;
40             }
41         }
42     }
43 }
44
45 void DebugPrinter::ejercer_tarea() {
46     atender_debug_msgs();
47 }
48
49 void DebugPrinter::print(const std::string &msg) {
50     debug_file << msg << std::flush;
51 }

```

May 24, 20 1:13

DebugPrinter.h

Page 1/1

```
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_BUFFFSIZE];
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
```

May 25, 20 10:12

EspecialistaMasaMadre.cpp

Page 1/2

```

1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #include <iostream>
6  #include "EspecialistaMasaMadre.h"
7
8
9  EspecialistaMasaMadre::EspecialistaMasaMadre() {
10     canal_cocineros.abrir();
11 }
12
13 EspecialistaMasaMadre::~EspecialistaMasaMadre() {
14     canal_cocineros.cerrar();
15     canal_cocineros.eliminar();
16     cerrar_canales_particulares();
17 }
18
19 void EspecialistaMasaMadre::ejercer_tarea() {
20     bool seguir_recibiendo_pedidos = true;
21     while (sigint_handler.getGracefulQuit() == 0 and seguir_recibiendo_pedidos)
22     {
23         std::string id_solicitante = escuchar_pedidos(
24             &seguir_recibiendo_pedidos);
25         if (seguir_recibiendo_pedidos) {
26             mandar_msj_debug("Especialista: recibí pedido de ración de"
27                             " MM de parte de: " + id_solicitante);
28             entregar_pedido(id_solicitante);
29         }
30     }
31
32     std::string EspecialistaMasaMadre::escuchar_pedidos(bool *
33     seguir_recibiendo_pedidos) {
34         ssize_t bytesLeídos = 0;
35         std::string id_solicitante_actual;
36
37         while (bytesLeídos < FIFO_ESPECIALISTA_MM_BUFFSIZE) {
38             bytesLeídos += canal_cocineros.leer(static_cast<void *>(buffer),
39                                                 FIFO_ESPECIALISTA_MM_BUFFSIZE);
40             std::string mensaje = buffer;
41             mensaje.resize(bytesLeídos);
42             if (bytesLeídos == 0) {
43                 *seguir_recibiendo_pedidos = false;
44                 break;
45             }
46             id_solicitante_actual.append(mensaje);
47         }
48         return id_solicitante_actual;
49     }
50
51     void EspecialistaMasaMadre::entregar_pedido(const std::string &id_solicitante) {
52         obtener_canal_envio_mm(id_solicitante)->escribir(
53             MENSAJE_PARA_ENVIAR_MM_A_COCINEROS, LENGTH_MSJ_ENVIO_MM);
54     }
55
56     bool
57     EspecialistaMasaMadre::existe_canal_envio_mm(const std::string &id_pedido) {
58         for (auto &it : canales_envio_mm) {
59             if (it.first == id_pedido) {
60                 return true;
61             }
62         }
63         return false;
64     }
65
66     FifoEscritura *EspecialistaMasaMadre::obtener_canal_envio_mm(
67         const std::string &id_solicitante) {
68         if (not existe_canal_envio_mm(id_solicitante)) {
69             canales_envio_mm.push_back(std::make_pair(id_solicitante,
70                                                         new FifoEscritura(
71                                                             id_solicitante)));
72             canales_envio_mm.back().second->abrir();

```

May 25, 20 10:12

EspecialistaMasaMadre.cpp

Page 2/2

```

73         return canales_envio_mm.back().second;
74     }
75     for (auto &it : canales_envio_mm) {
76         if (it.first == id_solicitante)
77             return it.second;
78     }
79 }
80
81 void EspecialistaMasaMadre::cerrar_canales_particulares() {
82     for (auto &it : canales_envio_mm) {
83         it.second->cerrar();
84         it.second->eliminar();
85         delete it.second;
86     }
87     canales_envio_mm.clear();
88 }

```


May 24, 20 1:12

EspecialistaMasaMadre.h

Page 1/1

```

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

```

May 24, 20 1:14

EventHandler.h

Page 1/1

```
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_ */
```

May 24, 20 1:10

Fifo.cpp

Page 1/1

```
1  #include "Fifo.h"
2
3  Fifo::Fifo(const std::string nombre) : nombre(nombre), fd(-1) {
4      mknod(static_cast<const char *>(nombre.c_str()), S_IFIFO | 0666, 0);
5  }
6
7  Fifo::~Fifo() {
8  }
9
10 void Fifo::cerrar() {
11     close(fd);
12     fd = -1;
13 }
14
15 void Fifo::eliminar() const {
16     unlink(nombre.c_str());
17 }
18
19 int Fifo::get_fd() {
20     return fd;
21 }
```

May 24, 20 1:10

FifoEscritura.cpp

Page 1/1

```
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 }
```

May 24, 20 1:11

FifoEscritura.h

Page 1/1

```
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_ */
```

May 20, 20 15:30

Fifo.h

Page 1/1

```
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_ */
```

May 24, 20 1:11

FifoLectura.cpp

Page 1/1

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

May 24, 20 1:11

FifoLectura.h

Page 1/1

```
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_ */
```


May 24, 20 1:15

main.cpp

Page 1/2

```

1  #include <iostream>
2  #include <fstream>
3  #include <cstring>
4  #include "utils/ProcessManager.h"
5
6  #define ARCHIVO_CONFIG_PATH "../config.cb"
7
8  void leer_config_file(int *cant_panaderos, int *cant_pizzeros,
9                      int *cant_recepcionistas);
10
11 bool print_debug_msgs(int argc, char **argv);
12
13 int main(int argc, char *argv[]) {
14     int cant_panaderos, cant_pizzeros, cant_recepcionistas;
15     leer_config_file(&cant_panaderos, &cant_pizzeros, &cant_recepcionistas);
16
17     std::cout << "Bienvenido a ConcuBread!" << std::endl;
18     std::cout << "Simulando..." << std::endl;
19
20     const std::unique_ptr<Proceso> &proceso_generado =
21         ProcessManager::crear_procesos(
22             cant_panaderos, cant_pizzeros,
23             cant_recepcionistas,
24             print_debug_msgs(argc, argv));
25     proceso_generado->ejercer_tarea();
26     return 0;
27 }
28
29 bool print_debug_msgs(int argc, char **argv) {
30     if (argc > 1) {
31         if (std::strcmp(argv[1], "--print-debug") == 0) {
32             if (std::strcmp(argv[2], "false") == 0) {
33                 std::cout << "Setting debug printing off..." << std::endl;
34                 return false;
35             }
36             if (std::strcmp(argv[2], "true") == 0) {
37                 std::cout << "Setting debug printing on..." << std::endl;
38                 return true;
39             }
40         }
41         std::cout << "Unidentified arg name. Debug printing set on as default."
42             << std::endl;
43         return true;
44     }
45     /* si no se especifica nada... */
46     return true;
47 }
48
49 void leer_config_file(int *cant_panaderos, int *cant_pizzeros,
50                     int *cant_recepcionistas) {
51     std::ifstream config_file(ARCHIVO_CONFIG_PATH);
52     if (config_file.is_open()) {
53         std::string line;
54         try {
55             std::getline(config_file, line);
56             *cant_panaderos = std::stoi(line);
57
58             std::getline(config_file, line);
59             *cant_pizzeros = std::stoi(line);
60
61             std::getline(config_file, line);
62             *cant_recepcionistas = std::stoi(line);
63         } catch (std::exception const &ex) {
64             std::cout
65                 << "Error reading config file."
66                 << " Please check config file format and try again."
67                 << std::endl;
68             config_file.close();
69             exit(0);
70         }
71         config_file.close();
72     } else {
73         std::cout

```

May 24, 20 1:15

main.cpp

Page 2/2

```

74         << "No config file found!"
75         << " Please, verify the config file is present and try again."
76         << std::endl;
77         exit(0);
78     }
79 }

```

May 24, 20 1:13

Padre.cpp

Page 1/1

```
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 }
```

May 21, 20 10:20

Padre.h

Page 1/1

```
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
```

May 24, 20 1:12

Panadero.cpp

Page 1/1

```
1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #include <iostream>
6  #include "Panadero.h"
7
8  Panadero::Panadero(int id_panadero) {
9      this->canal_recepcionista = std::make_unique<FifoLectura>(
10          ARCHIVO_FIFO_PANADEROS);
11      canal_recepcionista->abrir();
12
13      set_id(id_panadero);
14  }
15
16  void Panadero::set_id(int id_panadero) {
17      id = "Pan";
18      if (id_panadero < 10) {
19          id.append("0");
20      }
21      id.append(std::to_string(id_panadero));
22  }
23
24  Panadero::~Panadero() {
25      canal_recepcionista->cerrar();
26  }
```

May 24, 20 1:13

Panadero.h

Page 1/1

```
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
```

May 24, 20 1:13

Pizzero.cpp

Page 1/1

```
1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #include <iostream>
6  #include "Pizzero.h"
7
8  Pizzero::Pizzero(int id_pizzero) {
9      this->canal_recepcionista = std::make_unique<FifoLectura>
10         (ARCHIVO_FIFO_PIZZEROS);
11      canal_recepcionista->abrir();
12
13      set_id(id_pizzero);
14  }
15
16  Pizzero::~Pizzero() {
17      canal_recepcionista->cerrar();
18  }
19
20  void Pizzero::set_id(int id_pizzero) {
21      id = "Piz";
22      if (id_pizzero < 10) {
23          id.append("0");
24      }
25      id.append(std::to_string(id_pizzero));
26  }
```

May 24, 20 1:13

Pizzero.h

Page 1/1

```
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
```

May 13, 20 11:40

Proceso.cpp

Page 1/1

```
1  //  
2  // Created by rozanecm on 5/13/20.  
3  //  
4  
5  #include "Proceso.h"
```


May 24, 20 1:14

Proceso.h

Page 1/1

```
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
```

May 24, 20 1:15

ProcessManager.cpp

Page 1/1

```

1  //
2  // Created by rozanecm on 5/6/20.
3  //
4
5  #include "ProcessManager.h"
6  #include "../Procesos/DebugPrinter.h"
7  #include "../Procesos/Actores/Panadero.h"
8  #include "../Procesos/Actores/Pizzero.h"
9  #include "../Procesos/Actores/Recepcionista.h"
10 #include "../Procesos/Actores/EspecialistaMasaMadre.h"
11 #include "../Procesos/Padre.h"
12 #include <unistd.h>
13 #include <bits/unique_ptr.h>
14
15 std::unique_ptr<Proceso>
16 ProcessManager::crear_procesos(int cant_panaderos, int cant_pizzeros,
17                               int cant_recepcionistas, bool print_debug_msgs) {
18     /* Creacion panaderos */
19     for (int i = 0; i < cant_panaderos; ++i) {
20         if (fork() == 0) {
21             return std::make_unique<Panadero>(i);
22         }
23     }
24     /* Creacion pizzeros */
25     for (int i = 0; i < cant_pizzeros; ++i) {
26         if (fork() == 0) {
27             return std::make_unique<Pizzero>(i);
28         }
29     }
30     /* Creacion recepcionistas */
31     for (int i = 0; i < cant_recepcionistas; ++i) {
32         if (fork() == 0) {
33             return std::make_unique<Recepcionista>(i);
34         }
35     }
36     /* Creacion espec. masa madre */
37     if (fork() == 0) {
38         return std::make_unique<EspecialistaMasaMadre>();
39     }
40     /* Creacion Debug Printer */
41     if (fork() == 0) {
42         return std::make_unique<DebugPrinter>(print_debug_msgs);
43     }
44     return std::make_unique<Padre>();
45 }

```

May 24, 20 1:15

ProcessManager.h

Page 1/1

```
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
```

May 25, 20 10:13

Recepcionista.cpp

Page 1/2

```

1  //
2  // Created by rozanecm on 5/12/20.
3  //
4
5  #include <iostream>
6  #include <random>
7  #include "Recepcionista.h"
8
9  #define CANT_PIZZAS 2
10 #define CANT_PANES 1
11
12 Recepcionista::Recepcionista(int id) : id(id) {
13     canal_pizzeros.abrir();
14     canal_panaderos.abrir();
15 }
16
17 Recepcionista::~Recepcionista() {
18     canal_pizzeros.cerrar();
19     canal_panaderos.cerrar();
20     canal_pizzeros.eliminar();
21     canal_panaderos.eliminar();
22 }
23
24 void Recepcionista::ejercer_tarea() {
25     hacer_pedidos();
26 }
27
28 void Recepcionista::hacer_pedidos() {
29     /* La parte random esta sacada de
30      * https://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution
31      */
32     std::random_device rd;
33     std::mt19937 gen(rd());
34     std::uniform_int_distribution<> dis(1, 6);
35
36     auto cant_pizzas_a_encargar = dis(gen);
37     auto cant_panes_a_encargar = dis(gen);
38
39     auto decision_pizza_o_pan = dis(gen);
40
41     int pizzas_encargadas = 0;
42     int panes_encargados = 0;
43     while (sigint_handler.getGracefulQuit() == 0 and
44           (pizzas_encargadas < cant_pizzas_a_encargar or
45            panes_encargados < cant_panes_a_encargar)) {
46         if (decision_pizza_o_pan % 2 == 0 and
47             pizzas_encargadas < cant_pizzas_a_encargar) {
48             pizzas_encargadas++;
49             pedir_pizza(pizzas_encargadas, cant_pizzas_a_encargar);
50             sleep(dis(gen));
51         } else if (panes_encargados < cant_panes_a_encargar) {
52             panes_encargados++;
53             pedir_pan(panes_encargados, cant_panes_a_encargar);
54             sleep(dis(gen));
55         }
56         decision_pizza_o_pan = dis(gen);
57     }
58 }
59
60 void Recepcionista::pedir_pizza(int numero_de_encargo_pizza,
61                                int cant_total_de_pizzas_a_hacer) {
62     hacer_pedido_pizzeros(numero_de_encargo_pizza,
63                           cant_total_de_pizzas_a_hacer);
64 }
65
66 void Recepcionista::pedir_pan(int numero_de_encargo_pan,
67                               int cant_total_de_panes_a_hacer) {
68     hacer_pedido_panaderos(numero_de_encargo_pan, cant_total_de_panes_a_hacer);
69 }
70
71 void Recepcionista::hacer_pedido_pizzeros(int numero_de_encargo_pizza,
72                                            int cant_total_de_pizzas_a_hacer) {
73     auto id_pizza = id_prox_pizza();

```

May 25, 20 10:13

Recepcionista.cpp

Page 2/2

```

74
75     mandar_msj_debug("Recepc. " + std::to_string(id) +
76                     " Encargando pizza " + id_pizza + "(" +
77                     std::to_string(numero_de_encargo_pizza) + "/" +
78                     std::to_string(cant_total_de_pizzas_a_hacer) + ")");
79
80     mandar_msj_fifo(id_pizza, &canal_pizzeros);
81     cantidad_pizzas_encargadas++;
82 }
83
84 void Recepcionista::hacer_pedido_panaderos(int numero_de_encargo_pan,
85                                             int cant_total_de_panes_a_hacer) {
86     auto id_pan = id_prox_pan();
87
88     mandar_msj_debug("Recepc. " + std::to_string(id) +
89                     " Encargando pan " + id_pan + "(" +
90                     std::to_string(numero_de_encargo_pan) + "/" +
91                     std::to_string(cant_total_de_panes_a_hacer) + ")");
92
93     mandar_msj_fifo(id_pan, &canal_panaderos);
94     cantidad_panes_encargados++;
95 }
96
97 std::string Recepcionista::id_prox_pizza() const {
98     std::string id_pedido = "Piz";
99     if (this->id < 10) {
100         id_pedido += "0";
101     }
102     id_pedido += std::to_string(this->id);
103     if (this->cantidad_pizzas_encargadas < 10) {
104         id_pedido += "0";
105     }
106     id_pedido += std::to_string(cantidad_pizzas_encargadas);
107     return id_pedido;
108 }
109
110 std::string Recepcionista::id_prox_pan() const {
111     std::string id_pedido = "Pan";
112     if (this->id < 10) {
113         id_pedido += "0";
114     }
115     id_pedido += std::to_string(this->id);
116     if (this->cantidad_panes_encargados < 10) {
117         id_pedido += "0";
118     }
119     id_pedido += std::to_string(cantidad_panes_encargados);
120     return id_pedido;
121 }

```

May 24, 20 1:13

Recepcionista.h

Page 1/1

```

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

```

May 24, 20 1:14

SIGINT_Handler.h

Page 1/1

```
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_ */
```

May 24, 20 1:14

SignalHandler.cpp

Page 1/1

```

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 }

```

May 24, 20 1:14

SignalHandler.h

Page 1/1

```
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_ */
```