UNIVERSIDADE D
COIMBRA

Faculty of Sciences and Technology

OPERATIVE SYSTEMS - BACHELOR'S DEGREE IN INFORMATICS ENGINEERING

OFFLOADING SIMULATOR

Project by:
Nuno Batista, uc2022216127
Miguel Castela, uc2022212972

May, 2024

Table of Contents

1 - System Components

1.1 - system_manager.c			
int main(int argc, char* argv[])	Call system initializer		
1.2 - system_initialization.c			
int initialize_system(char*config_file)	Initializes the entire system using a a given configuration file.	**int** read_config_file(char* filename)	Reads the configuration file specified for system
1.3 - structures_creation.c			
int create_monitor_engine()	Creates the monitor engine process	**int** create_monitor_engine()	Creates the auxiliary shared memory semaphore and the engines semaphore
int create_shared_memory()	Creates the shared memory and auxiliary shared memory	**int** create_fifo_queues()	Creates the video and other queues
int create_auth_manager()	Creates and opens the named pipes, called by the ARM	**int** create_semaphores()	Creates the log semaphore and the shared memory semaphore
int create_message_queue()	Creates the message queue	**int** create_auth_engines()	Creates the initial auth engines
int create_pipes()	Creates the ARM process, which will create the auth engines and the receiver/sender threads		
1.4 – arm_threads.c			
void sender_thread()	Reads messages from queues, checks for auth engines, sends messages, and deactivates extra AE	**void** parse_and_send(char* message)	Sends messages to queues and deploys an extra engine if a queue is full
void deploy_extra_engine()	Deploys extra auth engine if both queues are full	**void** remove_extra_engine	Removes the extra Authorization Engine
void receiver_thread()	Listens to named pipes, sends messages to queues, and deploys an extra auth engine if queues are full.		
1.5 - auth engine.c			
void* sender_thread()	Reads messages from queues, searches for AEs, sends messages, and deactivates extra engines.	**void*** receiver_thread()	Listens to named pipes, sends messages to queues, and deploys an extra auth engine if queues are full.
void deploy_extra_engine()	Deploys extra auth engine if both queues are full.	**void** parse_and_send(char message)	Sends messages to queues and deploys an extra engine if a queue is full.
1.6 - monitor_engine.c			
void monitor_engine_process()	Monitor Engine main function, will tell users once they have reached 80%, 90% 100% of their plafond	**int** notify_user(int user_id, int percentage)	Notifies user about the amount of data. spent
void* periodic_notifications_thread()	Send notifications to the backoffice user in intervals of 30s	**int** deactivate_user(int user_id, int user_index)	Deactivates a user, called by the monitor. engine
1.7 - clean_up.c			
void clean_up()	Cleans up the system, called by the signal handler	**void** clean_up_arm()	Cleans up the ARM process and its structures
void notify_arm_threads()	Ask ARM threads to exit	**void** signal_handler(int signal)	Signal handler for SIGINT
1.8 - general_functions.c			
void write_to_log(char* message)	Writes a message to the log file	**unsigned long long** get_time_millis()	Prints the queues progress
void sleep_milliseconds(int milliseconds)	Sleeps for the specified amount of milliseconds	**void** print_progress(int current, int max)	Gets the current timestamp in milliseconds
void print_shared_memory()	Prints the current state of the shared memory		
1.9 - mobile_user.c			
int main(int argc, char *argv[])	Program's main entry point, manages initialization and work.	**void *** send_requests(void *arg)	Thread function for sending continuous. requests based on arguments.
void sleep_milliseconds(int milliseconds)	Suspends the current thread for a specified duration in milliseconds.	**int** is_positive_interger(char* str)	Checks if a string represents a positive integer.
void* message_receiver()	Thread function to continuously receive and process messages.	**void** clean_up()	Performs cleanup tasks before program exit.
void signal_handler(int sig)	Handles SIGINT signal for termination.	**int** send_initial_request(int initial_plafond)	Sends an initial registration request with a plafond value.
void print_arguments(int initial_plafond, int requests_left, int delta_vide int delta_music, int delta_social,int data_ammount)		Sends an initial registration request with a plafond value.	
1.10 – backoffice_user.c			
int main()	Initializes components, manages user commands, and controls the program's lifecycle.		
void interpret_command(char *command)	Interprets user commands, sends messages to the system via a named pipe.	**void** clean_up()	Performs cleanup tasks before exiting, like stopping threads, closing resources, and removing lockfiles.
void signal_handler(int signal)	Handles SIGINT to exit by calling clean_up().	**void** send_message(char *messag	Continuously processes messages from a queue until instructed to stop.
void* receiver()	Continuously processes messages from a queue until instructed to stop.	**void** print_statistics(char *messag	Initializes components, manages user commands, and controls the program's lifecycle.

2 - Summary and Main Features

Note: This is **highly** Summarized due to space constraints.

2.1 Initialization

This program starts by reading the configuration file and storing the correct parameters. It then calls the initializer function, creating the semaphores, the shared memory segments, the message queue, the names pipes, the monitor engine and the ARM.

The ARM will then create the sender and receiver threads, as well as the authorization engines and their respective unnamed pipes.

The monitor engine, on the other hand, will just create a thread for the back
office user's periodic notifications.

The program is now ready to receive the user's input and wait for a SIGINT signal to terminate.

2.2 Authentication Request Manager

The sender thread is now waiting for a notification from the receiver thread which will be sent once it has received a request from a user and sent it to one of the two queues.

Once notified, the sender thread will wait until an authorization engine is available for the request to be assigned to, this is done via the engines_sem, whose value is the number of currently available engines. If the receiver thread fills up one of the queues, the extra auth engine is deployed, and if the sender thread empties both of the queues to less than half of their capacity, the extra auth engine is removed.

2.3 Authorization Engines

The auth engine's work loop is simple, it waits for a message to arrive to its unnamed pipe, once it does, it parses the message accordingly and sends a notification to the monitor engine for it to check the state of the shared memory.

2.4 Monitor Engine

This process waits for a notification from the auth engines, once it receives it, it checks the shared memory for the current state of the system and sends it to the backoffice user if requested or it sends a notification to the mobile users informing them of their already spent plafond. This is done via the message queue. Moreover, there's a thread that sends a periodic notification to the backoffice user every 30 seconds with the current state of the system.

2.5 Mobile User

The mobile user is responsible for sending different types of periodic requests to
the ARM via the USER_PIPE with 3 threads, one for each type of request.

There's also another thread that waits for notifications from the system.

2.6 Backoffice User

The backoffice user can send two types of requests to the system via the BACK_PIPE, one to check the current state of the system and another to reset the shared memory stats.

2.7 Clean Up

When the system receives a SIGINT signal, every process will ignore it except for the system manager, the main process.

The system manager will then send a SIGTERM signal to the ARM process, which will terminate the receiver and the sender threads.

Once they are terminated, the ARM will send a SIGTERM to every auth engine, they will then finish processing their current requests and exit. When they exit, the ARM will also exit. Now, the system manager sends a SIGTERM to the monitor engine, which will also terminate.
Finally, the system manager will then clean the IPC resources, free the allocated memory and terminate.

Time spent on the project - 10h per student