Sistemas Operativos 2022/2023

Relatório Trabalho Prático

Projeto "Simulador para Internet das Coisas em contexto habitacional"

Gonçalo Senra - 2020213750 Rui Coelho - 2021235407

Ficheiros

sys_header.h - Contém todas as bibliotecas necessárias, define as constantes. Além disso, contém todas as estruturas como memória partilhada e internal Queue.

sensor.c - É o ficheiro relativo ao processo sensor, gera valores aleatórios para simular o trabalho de um sensor.

user_console.c - Corresponde à consola do utilizador.

worker.c - Este ficheiro contém o processo worker, este recebe tarefas do dispatcher através do unnamed pipe e as processa, caso a tarefa tenha vindo de um utilizador, envia uma mensagem ao mesmo através de uma message queue.

alerts_watcher.c - Este ficheiro contém o processo worker, este é "acordado" quando os dados de um sensor são atualizados (através de uma cond_var). Este, verifica se existe algum alerta a enviar, e se houver, envia-o ao user que criou o alerta, através da message queue.

home_iot.c - É o ficheiro que contém o processo principal (system manager), as suas threads(console_reader, sensor_reador, dispatcher). É aqui que as configurações do ficheiro passado por parâmetro são lidas e aplicadas, e consequentemente são criados e inicializados os blocos de memória partilhada, pipes, semáforos, message queue, internal Queue e cond_var. Para além disso, é neste ficheiro que estão as funções que criam todos os processos e threads, "signal handlers", e escrevem no *log file.*

Internal Queue

A internal queue pode receber dois tipos de mensagens: user messages; sensor messages.

Tal como especificado no enunciado, as mensagens do user são prioritárias. Inicialmente pensamos em colocar uma flag em cada nó para identificar a prioridade da mensagem, no entanto concluímos que uma melhor abordagem seria inserir as mensagens dos sensores no fim da fila e as mensagens do user no início. Por uma questão de cronologia, as mensagens que chegavam do console pipe eram colocadas atrás do último nó referente ao user.

Sincronização

Relativamente à sincronização, utilizamos dois tipos de semáforos: posix, pthread_mutex e cond_var.

- mutex_shm (posix) [1]: este semáforo é responsável por garantir que a memória partilhada não é corrompida, por múltiplos acessos concorrentes.

- mutex_log (posix) [2]: este semáforo é responsável por garantir que o ficheiro log não é corrompido.
- active_workers (posix) [3]: se não houver nenhum processo worker disponível a thread dispatcher espera por que termine a sua tarefa para lhe puder enviar outra.
- mutex_queue (pthread_mutex) [4]: este mutex controla os acessos concorrentes à internal queue, para que a mesma não se corrompa.
- sem_qsize (posix) e sem_qcons (posix) [5, 6]: estes semáforos são utilizados como consumidor/produtor, o sem_qsize indica o número de posições vazias da internal queue, e o sem_qcons indica o número de posições preenchidas.
- mutex_cond (pthread_mutex) e sens_watcher (cond_var) [7, 8]: este mutex e variavel de condição são responsáveis por acordar o alerts_watcher quando os dados de um sensor são atualizados.

Conclusão

Achamos que o tema do projeto foi bastante interessante, já que o mesmo se pode enquadrar num problema do mundo real. Para além disso, este projeto permitiu consolidar o conhecimento adquirido nas aulas teóricas e práticas-laboratoriais.

Para este projeto despendemos um total de 120h (Gonçalo Senra – 60h, Rui Coelho – 60h).

Arquitetura do programa