Sistemas Operativos 2017/2018

Trabalho Prático

Gestão de Urgências

1. Objetivos do trabalho

- Desenvolver um simulador das urgências de um hospital na linguagem de programação C. Este simulador será capaz de representar os processos de triagem e de atendimento que decorrem no hospital.

- Explorar mecanismos de gestão de processos, *threads*, comunicação e sincronização em Linux.

2. Visão geral do funcionamento da aplicação

A Figura 1 apresenta uma visão geral do funcionamento do sistema a implementar no contexto do Trabalho Prático.

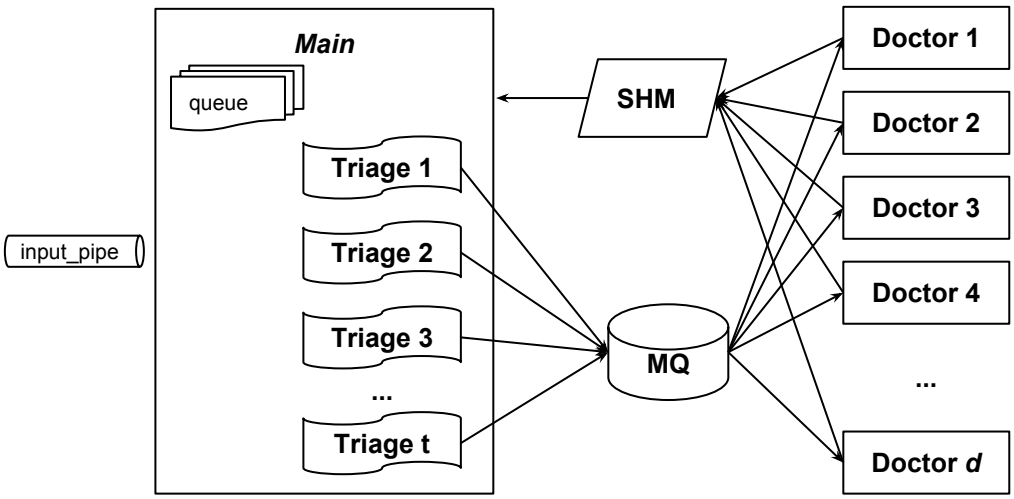

Figure 1 - Visão geral do sistema a ser implementado.

Tal como a figura anterior ilustra, o sistema é baseado em vários processos. Estes processos são responsáveis pelas seguintes funcionalidades:

- O processo principal é responsável por iniciar todo o sistema. Nomeadamente, deve criar o *named pipe* ("input_pipe"), a fila de mensagens ("MQ"), a zona de memória partilhada ("SHM"), as *threads* de triagem ("Triage x") e os processos doutores ("Doctor x"). Deve também ler

estatísticas a partir da memória partilhada para apresentar ao utilizador. Deve receber os dados dos pacientes de um *named pip*e e colocá-los na fila de triagem ("queue").

- Os processos doutor devem obter os processos dos pacientes da fila de mensagens, atendendo-os de acordo com a prioridade atribuída na triagem.

A seguir explicam-se em maior detalhe as funcionalidades a implementar na aplicação.

3. Descrição das funcionalidades a implementar

3.1. Recepção de pacientes

O processo principal deve ficar à escuta em um *named pipe* de nome "input_pipe" pela chegada de novos pacientes. Quando novos clientes chegam, devem ser adicionados a uma fila, de forma a serem triados pela ordem de chegada.

São aceites os dois formatos de entrada de pacientes: um paciente específico ou um grupo de pacientes. No caso dos pacientes específicos, estes devem ser identificados pelo seu nome próprio. Para cada paciente (ou conjunto de pacientes) deve ser definida a quantidade de tempo necessário para a triagem e para o atendimento. Este valor será usado nos passos subsequentes.

Deve ser usado um formato simples para envio da informação pelo pipe, de acordo com o exemplo seguinte. Tal como o exemplo ilustra, podemos identificar um paciente pelo nome ou, em alternativa, um grupo de pacientes (situação em que o programa deve atribuir automaticamente nomes ou números aos vários pacientes desse grupo - e.g. "20171201-001").

Exemplo de conteúdo a enviar pelo *named pipe*:

```
João 10 50 1 // Paciente João: 10ms triagem, 50ms atendimento, prioridade 1
8    10 65 3 // 8 pacientes: 10ms de triagem, 65ms atendimento, prioridade 3
```

Cada paciente deve ser representado internamente por uma estrutura, que deverá guardar o seu número de chegada, o seu nome e os dados necessários para calcular todas as estatísticas mencionadas no restante enunciado. Por exemplo, aquando da recepção dos pacientes, o instante inicial deve ser guardado.

3.2. Triagem dos pacientes

O número de *threads* disponíveis para esta tarefa é definida pelo ficheiro de configuração (parâmetro "TRIAGE", ver 3.5) e pode ser alterada manualmente pelo administrador para ajustar a capacidade da triagem ao ritmo de chegada de paciente. Essa alteração durante a execução é feita enviando pelo *named pipe* um

comando especial de alteração do parâmetro "TRIAGE" (ex: TRIAGE=10). O início e fim de cada *thread* deve ser escrito no ficheiro de *log*.

Cada *thread* deve começar por tirar um paciente da fila e registar o evento no log do sistema. De seguida efectua o processo de triagem, que consiste em atribuir ao paciente uma prioridade (definida nos dados recebidos pelo *pipe*). Essa operação demora o tempo definido nos dados originalmente recebidos pelo *named pipe* e é variável de paciente para paciente. Após a prioridade atribuída, o paciente deve ser inserido na fila de mensagens destinada aos processos doutores, para que possa ser mais tarde atendido. No fim do processo de triagem estar concluído deve inserir novo registo no *log*.

A *thread* deve ainda escrever na zona de memória correspondente às estatísticas toda a informação necessária para computar os valores descritos em **3.4**.

O ficheiro de log deverá ser mapeado em memória (ser um *MMF, memory-mapped file*) por questões de performance. Deve reservar um ficheiro grande o suficiente para evitar a necessidade de voltar a mapear o ficheiro.

3.3. Atendimento pelos processos doutor

O atendimento dos pacientes é feito por um conjunto de processos "Doctor". Cada processo atua individualmente e obtém um paciente da fila de mensagens de cada vez. A priorização efectuada pela triagem deve ser respeitada, sendo que entre pacientes da mesma prioridade deve ser atendido aquele que estiver há mais tempo em espera.

O atendimento de cada paciente deve começar com o registo do mesmo no *log* do sistema, aguardar o intervalo de tempo predeterminado para o paciente, e novo registo do fim do atendimento. Por último, o processo doutor deve escrever na zona de memória correspondente às estatísticas, as informações necessárias para computar as estatísticas descritas em 3.4.

Cada um dos processos trabalha durante um período de tempo correspondente a um turno (parâmetro "SHIFT_LENGTH", ver 3.5), ao fim do qual termina o paciente que tem em mãos e sai. O processo principal deve detectar este evento, registá-lo no *log* e iniciar um novo processo doutor.

O número de processos doutores a trabalhar em simultâneo é definido pelo ficheiro de configuração (parâmetro "DOCTORS", ver 3.5). Este número não pode ser alterado manualmente. No entanto, um processo doutor temporário pode ser criado no caso de o número de pacientes em espera na fila de mensagens ultrapassar o máximo (parâmetro "MQ_MAX", ver 3.5), sendo que este processo deve terminar assim que o número de pacientes na fila descer abaixo dos 80% do máximo.

3.4. Informação estatística sobre pacientes atendidos

Pretendem-se manter estatísticas relativas ao funcionamento do sistema. Estas estatísticas devem ser mantidas em memória partilhada, de forma a que possam ser actualizadas tanto pela triagem como pelos doutores.

Tal como a Figura 1 ilustra, o processo principal é capaz de obter informação sobre os pedidos aceites e tratados pelo servidor, através da memória partilhada. Após a receção de um sinal do tipo SIGUSR1, o processo principal deverá escrever para o écran a seguinte informação estatística agregada:

- Número total de pacientes triados;
- Número total de pacientes atendidos;
- Tempo médio de espera antes do início da triagem;
- Tempo médio de espera entre o fim da triagem e o início do atendimento;
- Média do tempo total que cada utilizador gastou desde que chegou ao sistema até sair;

3.5. Arranque e terminação do Servidor

No seu arranque, o servidor deverá ler a configuração inicial do ficheiro "config.txt" que deverá conter os seguintes dados:

```
número de threads na triagem
número de processos doutor
duração do turno em segundos
tamanho máximo da fila para atendimento
```

Exemplo de um ficheiro de configuração:

```
TRIAGE=5
DOCTORS=10
SHIFT_LENGTH=5
MQ_MAX=20
```

De seguida, deverá criar as *threads* (triagem) e os processos necessários (doutor), bem como todos os restantes recursos de comunicação e sincronização necessários. O PID de cada um dos processos doutor deverá ser escrito no ecrã e no log sempre que ele inicia o seu turno e sempre que o termina.

O Servidor deverá estar igualmente preparado para terminar, após a receção, por parte do processo principal, de um sinal do tipo SIGINT. Nessa altura, o servidor deverá deixar de receber novos pedidos, aguardar a terminação de todos os pedidos pendentes, e só depois terminar os processos e fazer a limpeza no sistema de todos os recursos partilhados.

4. Checklist

Esta lista serve como indicadora das tarefas a realizar e assinala as componentes que serão objecto de avaliação na defesa intermédia.

Processo	**Tarefa**	**Avaliado na defesa intermédia?**
Processo "Principal" ("Main")	Arranque do servidor e aplicação das configurações existentes no ficheiro "config.txt"	S
	Criação de todos os processos "Doutor"	S
	Criação da memória partilhada	S
	Criação do *named pipe*	
	Criação da Fila de Mensagens	
	Criação da *pool* de *threads*	S
	Mapeamento em memória do ficheiro de *log*	
	Leitura correcta dos pacientes recebidos (individuais ou em grupo) pelo *named pipe* e colocação numa fila para triagem	
	Triagem dos pacientes de acordo com os dados recebidos de cada paciente e respeitando os tempos definidos	
	Escrita em memória partilhada das estatísticas de triagem	S (só prova de conceito)
	Escrita no ficheiro de *log* dos dados da triagem	
	Inserção de clientes triados na fila de mensagens	
	Alteração do número de *threads* através de comando enviado pelo *named pipe*	
	Escrever a informação estatística no ecrã como resposta ao sinal SIGUSR1	
Processos "Doutor" ("Doctor")	Tratamento correcto de cada paciente, incluindo o respeito pela prioridade e pelos tempos definidos	
	Escrita no ficheiro de *log* dos dados relativos ao atendimento	
	Escrita correta das estatísticas	S
	Criação de novos processos quando um termina, respeitando os tempos de turno	S
	Criação dinâmica de novos processos "Doutor" em caso de necessidade	S (só prova de conceito)
Geral	Detecção e tratamento de erros.	S
	Suporte de concorrência no tratamento de pedidos	
	Sincronização com mecanismos adequados (semáforos, *mutexes* ou variáveis de condição)	
	Prevenção de interrupções indesejadas por sinais e fornecer a resposta adequada aos vários sinais	
	Após recepção de SIGINT, terminação controlada de todos os processos e *threads,* e libertação de todos os recursos.	S (só prova de conceito)

Notas importantes

- **Não será tolerado plágio, cópia de partes de código entre grupos ou qualquer outro tipo de fraude**. Tentativas neste sentido resultarão na **classificação de ZERO valores** e na consequente **reprovação na cadeira**.
- Em vez de começar a programar de imediato pense com tempo no problema e estruture adequadamente a sua solução.
- Inclua na sua solução o código necessário à deteção e correção de erros.
- Evite esperas ativas no código e assegure a terminação limpa do servidor, ou seja, com todos os recursos utilizados a serem removidos.
- Utilize um *makefile* para simplificar o processo de compilação.
- Inclua informação de *debug* que facilite o acompanhamento da execução do programa, utilizando por exemplo a seguinte abordagem:

```
#define DEBUG //remove this line to remove debug messages
(...)
#ifdef DEBUG
printf("Creating shared memory\n");
#endif
```

- Todos os trabalhos deverão funcionar na student2.dei.uc.pt ou, em alternativa, na VM fornecida.
- A defesa final do trabalho é obrigatória para todos os elementos do grupo. A não comparência na defesa final implica a classificação de **ZERO valores** no trabalho.
- O trabalho pode ser realizado em grupos de até 2 alunos.

5. Metas, entregas e datas

Data	**Meta**	
Semana de 30/10/2017	Demonstração intermédia	Os alunos deverão apresentar o seu trabalho nas aulas PL. A demonstração deverá contemplar todos os pontos referidos na *checklist* que consta deste enunciado. A defesa intermédia valerá **20%** da cotação do projeto.
Data de entrega **Sem penalização: 10/12/2017-20h00** **Com penalização de 20%: 11/12/2017-06h00** **Encerramento das submissões: 11/12/2017-06h00**	Entrega final	O projeto final deverá ser submetido no InforEstudante, tendo em conta o seguinte: ● Os **nomes** e **números** dos alunos do grupo devem ser colocados no início dos ficheiros com o código fonte. ● Com o código deve ser entregue um **relatório** sucinto (no máximo 2 páginas A4), no formato **pdf**, que explique as opções tomadas na construção da solução. Inclua um esquema da arquitectura do seu programa. Inclua também informação sobre o tempo total despendido (pelos dois elementos do grupo) no projeto. ● Crie um arquivo no formato **ZIP** com todos os ficheiros do trabalho. ● Não serão admitidas entregas por e-mail. A defesa final valerá **80%** da cotação do projecto e consistirá numa análise detalhada do trabalho apresentado.
11/12/2017 a 19/12/2017	Defesa	Defesas em grupo nas aulas PL.