Sistemas Operativos

Projeto 2025/2026

Urgências@DEI

NOTA: Antes de começar o trabalho, leia o enunciado até ao fim!

1 Introdução

Neste trabalho, pretende-se desenvolver um simulador das urgências de um hospital usando a linguagem de programação C e mecanismos de IPC e de concorrência estudados. Este simulador será capaz de representar os processos de triagem e de atendimento que decorrem num hospital.

2 Descrição do sistema de simulação

Nesta secção será descrita a arquitetura técnica do sistema, incluindo as funcionalidades dos vários componentes do simulador e a forma como comunicam.

2.1 Estrutura do sistema

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

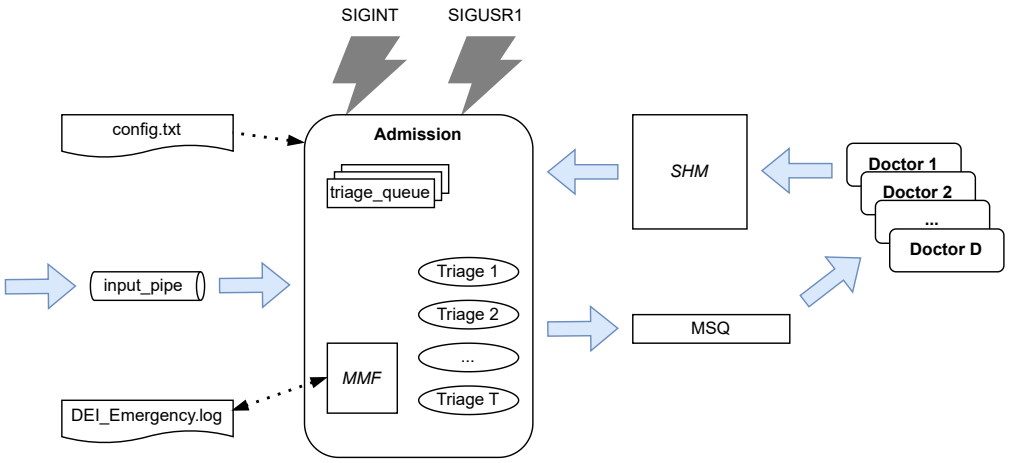

Figura 1: Simulador do processo de triagem

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

- O **processo *Admission*** é o processo com que o sistema arranca e é responsável por iniciar todo os processos, *threads* e recursos necessários. Nomeadamente, deve criar o *named pipe* ***input_pipe***, a fila de mensagens ***MSQ***, a zona de memória partilhada ***SHM***, as *threads* de triagem ***Triage x*** e os processos ***Doctor x***. Deve também ler estatísticas a partir da memória partilhada (***SHM***) para apresentar ao utilizador. Recebe dados dos pacientes a partir do *named pipe* ***input_pipe*** e coloca-os na fila de triagem ***triage_queue***.

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

2.2 Descrição das funcionalidades a implementar

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

2.2.1 Receção de pacientes

O processo ***Admission*** deve ficar à escuta num *named pipe* de nome ***input_pipe*** pela chegada de novos pacientes. Quando novos clientes chegam, devem ser adicionados à fila ***triage_queue***, de forma a serem triados pela ordem de chegada. Pode existir escrita concorrente no *named pipe*.

São aceites 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 *named pipe*, de acordo com o exemplo seguinte:

```
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
```

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. “20251201-001”).

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 neste enunciado. Por exemplo, na receção dos pacientes, o instante inicial de chegada deve ser guardado.

A fila ***triage_queue*** tem um número limite de posições (parâmetro “TRIAGE_QUEUE_MAX”, ver 2.2.5). Caso um novo pedido não tenha espaço suficiente na fila para ser colocado, deve ser eliminado e uma mensagem de erro deve ser escrita no *log*.

2.2.2 Triagem dos pacientes

O número de *threads* disponíveis para esta tarefa é definida no ficheiro de configuração (parâmetro “TRIAGE”, ver 2.2.5) 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 efetua o processo de triagem, que consiste em atribuir ao paciente uma prioridade (definida nos dados recebidos pelo *named pipe*). Essa operação demora o tempo definido nos dados originalmente recebidos e é variável de paciente para paciente. Após a prioridade atribuída, o

paciente deve ser inserido na fila de mensagens ***MSQ*** 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 calcular os valores descritos em 2.2.4.

O ficheiro de *log* usado para guardar informações produzidas pelo processo ***Admission*** e suas *threads*, deverá ser mapeado em memória (será 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.

2.2.3 Atendimento pelos processos *Doctor*

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 tendo em conta a priorização efetuada pela triagem, 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 da sua hora de entrada, aguardar o intervalo de tempo predeterminado para o paciente, e novo registo no fim do atendimento. Por último, o processo ***Doctor*** deve escrever na zona de memória correspondente às estatísticas, as informações necessárias para calcular as estatísticas descritas em 2.2.4.

Cada um dos processos trabalha durante um período correspondente a um turno (parâmetro "SHIFT_LENGTH", ver 2.2.5), ao fim do qual termina o paciente que tem em mãos e sai (Nota: também deverá sair caso não esteja com nenhum paciente na altura do fim do turno!). O processo principal deve detetar este evento, registá-lo no *log* e iniciar um novo processo ***Doctor***.

O número de processos ***Doctor*** a trabalhar em simultâneo é definido pelo ficheiro de configuração (parâmetro "DOCTORS", ver 2.2.5). Este número não pode ser alterado em tempo de execução. No entanto, um processo ***Doctor*** temporário pode ser criado no caso de o número de pacientes em espera na fila de mensagens ultrapassar o máximo (parâmetro "MSQ_WAIT_MAX", ver 2.2.5), sendo que este processo deve terminar assim que o número de pacientes à espera descer abaixo de 80% do máximo.

As informações sobre o início de atendimento de cada um dos clientes, o fim do atendimento e o fim do turno devem ser escritas no ecrã.

2.2.4 Informação estatística sobre pacientes atendidos

Pretendem-se manter estatísticas relativas ao funcionamento do sistema. Estas estatísticas devem ser mantidas em memória partilhada, de forma que possam ser atualizadas tanto pela triagem como pelos processos ***Doctor***. Tal como a Figura 1 ilustra, o processo ***Admission*** 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 ***Admission*** deverá escrever para o ecrã 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 paciente gastou desde que chegou ao sistema até sair;

2.2.5 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 máximo de posições na fila a aguardar triagem
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_QUEUE_MAX = 10
TRIAGE=5
DOCTORS=10
SHIFT_LENGTH=5
MSQ_WAIT_MAX=20
```

De seguida, deverá criar as *threads* (triagem) e os processos necessários, bem como todos os restantes recursos de comunicação e sincronização necessários. O PID de cada um dos processos ***Doctor*** 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 de todos os recursos partilhados.

2.2.6 Log da aplicação

Todo o *output* da aplicação deve ser escrito de forma legível num ficheiro de texto "DEI_Emergency.log". Cada escrita neste ficheiro deve ser sempre precedida pela escrita da mesma informação na consola, de modo a poder ser facilmente visualizada enquanto decorre a simulação.
Deverá pôr no ***log*** todos os eventos relevantes acompanhados da sua data e hora, incluindo:

- Data/hora da mensagem
- Início e fim do programa
- Criação de cada um dos processos e *threads*
- Inicio e fim da triagem
- Inicio e fim dos turnos
- Erros ocorridos
- Pacientes descartados
- Sinais recebidos

3 Checklist

Esta lista serve apenas como uma indicação das tarefas a serem realizadas e marca os componentes que serão avaliados na defesa intermédia. As tarefas com a indicação «(preliminar)» não precisam de ser totalmente concluídas na defesa intermédia.

Processo	**Tarefa**	**Avaliado na defesa intermédia?**
Processo ***Admission***	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 (preliminar)
	Escrita no ficheiro de *log* dos dados da triagem	
	Inserção de clientes triados na fila de mensagens	
	Alteração do nº de *threads* através de comando enviado pelo *named pipe*	
	Escrever a informação estatística no ecrã como resposta ao sinal SIGUSR1	
Processos ***Doctor***	Tratamento correto de cada paciente, incluindo o respeito pela prioridade e pelos tempos definidos	
	Cumprimento dos tempos de turno	
	Escrita correta das estatísticas	S (preliminar)
	Criação de novos processos, respeitando os tempos de turno	S (preliminar)
	Criação dinâmica de novos processos "Doutor" em caso de necessidade	S (preliminar)
Ficheiro *log*	Envio sincronizado do *output* para MMF e ecrã.	
Geral	Criar um *makefile*	
	Diagrama com a arquitetura e mecanismos de sincronização	S
	Suporte de concorrência no tratamento de pedidos	
	Deteção e tratamento de erros.	S
	Atualização da shm por todos os processos e *threads* que necessitem	S
	Sincronização com mecanismos adequados (semáforos, *mutexes* ou variáveis de condição)	
	Prevenção de interrupções indesejadas por sinais não especificados; fornecer a resposta adequada aos vários sinais especificados no enunciado.	
	Após receção de SIGINT, terminação controlada de todos os processos e *threads,* e libertação de todos os recursos.	S (preliminar)

4 Notas importantes

- Leia atentamente este enunciado e esclareça dúvidas com os docentes.
- Em vez de começar a programar de imediato pense com tempo no problema e estruture adequadamente a sua solução. Soluções mais eficientes e que usem menos recursos serão valorizadas.
- Inclua na sua solução o código necessário à deteção e correção de erros.
- Evite esperas ativas no código, sincronize o acesso aos dados sempre que necessário e assegure a terminação limpa do servidor, ou seja, com todos os recursos utilizados a serem removidos.
- **Penalizações:**
  - A não compilação do código enviado implica uma classificação de **ZERO valores** na meta correspondente.
  - Esperas ativas serão fortemente penalizadas! Use o comando *top* num terminal, de modo a assegurar que o programa não gasta mais CPU que o necessário!
  - Acessos concorrentes que, por não serem sincronizados, puderem levar à corrupção de dados, serão fortemente penalizados!
  - Uso da função *sleep* ou estratagemas similares para evitar problemas de sincronização, serão fortemente penalizados!
- 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 VM fornecida** ou, em alternativa, na máquina **student2.dei.uc.pt**.
- Compilação: o programa deverá compilar com recurso a uma *makefile*; não deve ter erros em qualquer uma das metas; evite também os *warnings*, exceto quando tiver uma boa justificação para a sua ocorrência (é raro acontecer!).
- A defesa final do trabalho é obrigatória e todos os elementos do grupo devem participar. 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 (grupos com apenas 1 aluno devem ser evitados e grupos com mais de 2 alunos não são permitidos).
- Os alunos do grupo devem pertencer a turmas PL do mesmo docente. Grupos com alunos de turmas de docentes diferentes são exceções que carecem de aprovação prévia.
- A nota da defesa é individual pelo que cada um dos elementos do grupo poderá ter uma nota diferente;
- Ambas as defesas, intermédia e final, devem ser realizadas na mesma turma e com o mesmo docente.
- **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**. Dependendo da gravidade poderão ainda levar a processos disciplinares.
- **Uso de ferramentas de inteligência artificial (IA):** não se recomenda a implementação direta através de ferramentas de IA. Independentemente da decisão dos alunos, durante a defesa serão feitas questões específicas sobre o código apresentado e podem ainda ser pedidas modificações pontuais. Não saber explicar detalhadamente as opções e código apresentados, ou não saber realizar alterações básicas ao código, poderá levar a fortes penalizações.
- Todos os trabalhos serão escrutinados para deteção de cópias de código.
- Para evitar cópias, os alunos **não podem** colocar código em repositórios de acesso público.

5 Metas, entregas e datas

Data	**Meta**	
Data de entrega no Inforestudante 24/11/2025, 17h	Entrega intermédia	Crie um arquivo no formato **ZIP** (**NÃO SERÃO ACEITES OUTROS FORMATOS**) com todos os ficheiros do trabalho e submeta-o no Inforestudante: ● Os **nomes** e **números** dos alunos do grupo devem ser colocados no início dos ficheiros com o código fonte. ● Inclua todos os ficheiros fonte e de configuração necessários e também um Makefile para compilação do programa. ● Não inclua quaisquer ficheiros não necessários para a compilação ou execução do programa (ex. diretórios ou ficheiros de sistemas de controlo de versões) ● Com o código deve ser entregue 1 página A4 no formato **pdf (NÃO SERÃO ACEITES OUTROS FORMATOS)**, que descreva a arquitetura e as opções de sincronização. ● **Não serão admitidas entregas por e-mail**.
Semana a começar em 24/11/2025	Demonstração/ defesa intermédia	● A demonstração deve abranger todos os pontos mencionados na *checklist*. ● A demonstração/defesa será realizada nas aulas de PL. ● A defesa intermédia vale **20%** da nota do projeto.
Data de entrega final no Inforestudante 15/12/2025	Entrega final	Crie um arquivo no formato **ZIP** (**NÃO SERÃO ACEITES OUTROS FORMATOS**) com todos os ficheiros do trabalho e submeta-o no Inforestudante: ● Os **nomes** e **números** dos alunos do grupo devem ser colocados no início dos ficheiros com o código fonte. ● Inclua todos os ficheiros fonte e de configuração necessários e também um Makefile para compilação do programa. ● Não inclua quaisquer ficheiros não necessários para a compilação ou execução do programa (ex. diretórios ou ficheiros de sistemas de controlo de versões) ● Com o código deve ser entregue um **relatório** sucinto (no máximo 2 páginas A4), no formato **pdf (NÃO SERÃO ACEITES OUTROS FORMATOS)**, que explique as opções tomadas na construção da solução. Inclua um esquema da arquitetura do seu programa. Inclua também informação sobre o tempo total despendido (por cada um dos dois elementos do grupo) no projeto. ● **Não serão admitidas entregas por e-mail**.
16/12/2025 a 19/12/2025	Defesa final	● A defesa final vale **80%** da cotação do projeto e consistirá numa análise detalhada do trabalho apresentado. Esta defesa poderá incluir uma parte escrita se necessário. ● As defesas serão feitas durante as aulas PL e em dias/horas extra a combinar. ● É necessária inscrição para a defesa.

Depois da submissão é aconselhado o *download* dos ficheiros, para verificar se tudo o que é necessário foi submetido. Apenas será considerado para avaliação o que for entregue.