Sistemas Operativos - Trabalho Prático

Sistema Integrado de Gestão Hospitalar

Ano Letivo: 2024/2025
Curso: Licenciatura/Mestrado em Engenharia Informática
Unidade Curricular: Sistemas Operativos
Tipo: Trabalho Individual
Prazo de Entrega: [consultar moodle]

PARTE 1 - DESCRIÇÃO DO TRABALHO

1. Objetivos do Trabalho

Este trabalho prático tem como objetivos principais aplicar de forma integrada todos os conceitos fundamentais de Sistemas Operativos aprendidos ao longo do ano letivo, através do desenvolvimento de um sistema complexo e realista de gestão hospitalar.

1.1 Objetivos Técnicos

- **Desenvolver** um sistema completo de gestão hospitalar com 4 componentes interdependentes
- **Aplicar** conceitos avançados de gestão de processos, threads e sincronização
- **Implementar** múltiplos mecanismos de comunicação inter-processos (IPC)
- **Garantir** sincronização correta em ambientes de alta concorrência
- **Gerir** recursos partilhados de forma eficiente e sem conflitos
- **Implementar** algoritmos de escalonamento otimizados

1.2 Conceitos de SO Cobertos

Conceito	Peso	Aplicação no Projeto
Gestão de Processos	20%	5 processos (1 central + 4 especializados) com fork() e hierarquia
Gestão de Threads	25%	Múltiplas threads por componente, sincronização complexa
IPC - Message Queues	8%	3 filas com prioridades diferentes
IPC - Shared Memory	8%	5 segmentos para estados e estatísticas
IPC - Pipes	4%	5 named pipes para comandos e comunicação
Sincronização	15%	Mutexes, semáforos, variáveis de condição, prevenção de deadlock
Gestão de Memória	10%	Alocação dinâmica, shared memory, prevenção de leaks
Sistema de Ficheiros	10%	Config, logs, resultados, estatísticas

---

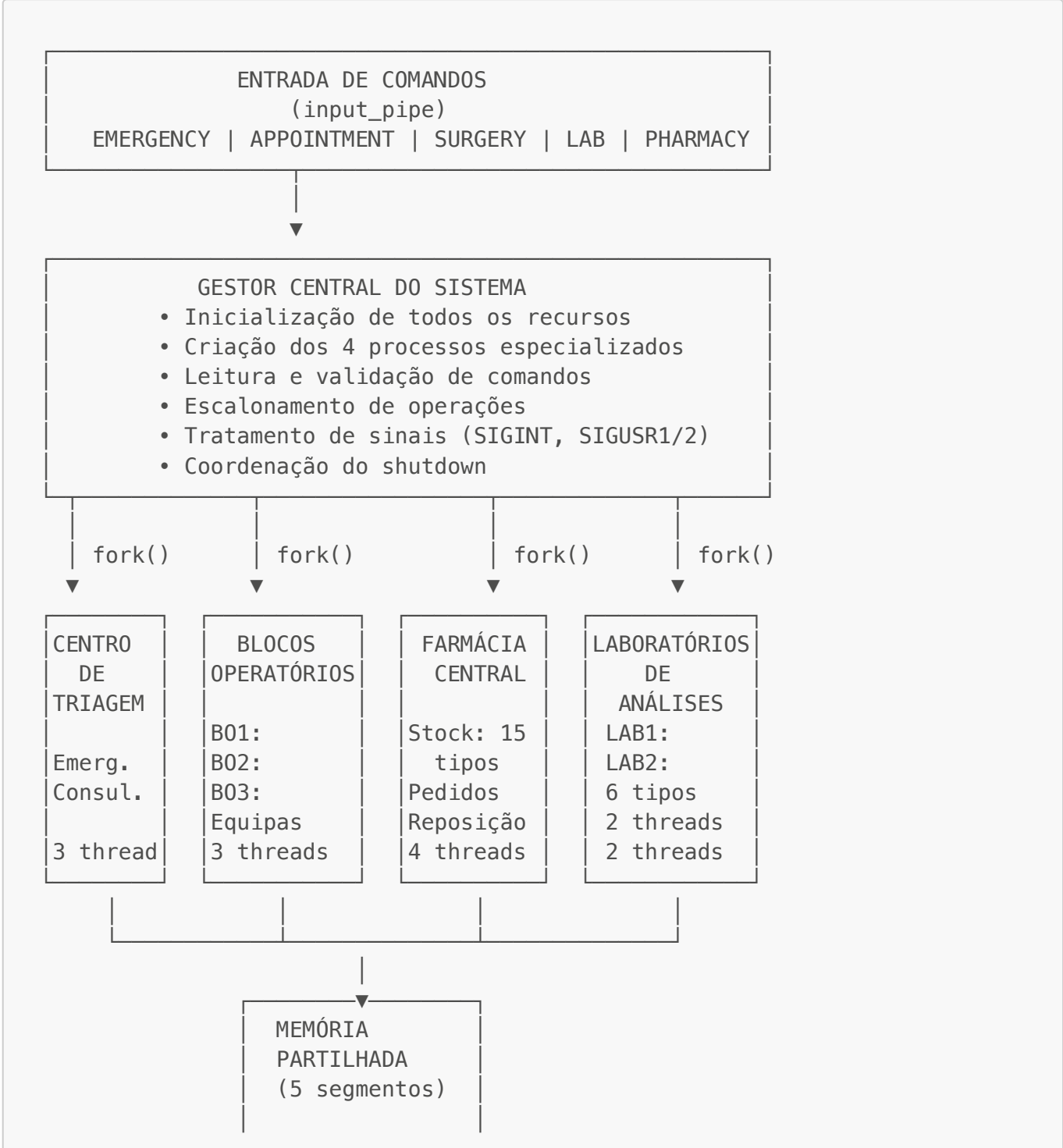
2. Contextualização e Descrição do Sistema

2.1 Visão Geral

Você foi contratado para desenvolver um **Sistema Integrado de Gestão Hospitalar** que coordene as operações críticas de um hospital de médio porte. O sistema é composto por **4 componentes principais** que devem funcionar em paralelo e comunicar entre si:

1. **Centro de Triagem** - Recebe e processa pacientes de emergência e consultas agendadas
2. **Blocos Operatórios** - Gere 3 salas cirúrgicas com especialidades distintas (Cardiologia, Ortopedia, Neurologia)
3. **Farmácia Central** - Processa pedidos de medicamentos e gere stock de 15 tipos diferentes
4. **Laboratórios de Análises** - Executa 6 tipos de exames em 2 laboratórios especializados

2.2 Arquitetura do Sistema



```
                    │ • Estatísticas  │
                    │ • Estado BO     │
                    │ • Stock Farm.   │
                    │ • Fila Labs     │
                    │ • Log Crítico   │
                    └─────────────────┘
```

2.3 Desafios do Sistema

- **Alta Concorrência**: Múltiplos pacientes e operações simultâneas
- **Recursos Limitados**: Salas cirúrgicas, equipas médicas, stock de medicamentos
- **Prioridades Complexas**: 5 níveis de triagem + urgências cirúrgicas
- **Dependências**: Cirurgias requerem análises prévias E medicamentos disponíveis
- **Sincronização Crítica**: Acesso concorrente a dados partilhados sem corrupção
- **Tempo Real**: Decisões rápidas para maximizar throughput

---

3. Componentes do Sistema - Descrição Detalhada

3.1 Gestor Central do Sistema (Processo Principal)

Função: Coordenação geral e inicialização do sistema.

Responsabilidades:

1. Ler ficheiro de configuração `config.txt` com todos os parâmetros
2. Criar todos os recursos IPC necessários:
    - 3 Message Queues (urgente, normal, respostas)
    - 5 Segmentos de Shared Memory
    - 5 Named Pipes
    - 7 Semáforos POSIX
3. Criar os 4 processos especializados usando `fork()`
4. Monitorizar continuamente o `input_pipe` para novos comandos
5. Validar sintaxe e lógica de comandos recebidos
6. Criar threads de pacientes/operações conforme comandos
7. Responder a sinais do sistema
8. Coordenar shutdown gracioso

Threads Criadas:

- Thread principal (main thread)
- Thread monitora de comandos
- Threads individuais por paciente/operação (criadas dinamicamente)

PID: Processo pai de todos os outros

3.2 Centro de Triagem (Processo Independente)

Função: Gestão de emergências e consultas programadas.

Responsabilidades:

1. Receber pacientes de emergência com 5 níveis de prioridade (1=mais crítico)
2. Gerir consultas agendadas
3. Monitorizar estabilidade vital dos pacientes (decresce 1 unidade/time unit)
4. Manter 2 filas ordenadas:
    - **Fila Emergências**: Por prioridade crítica > nível triagem > tempo
    - **Fila Consultas**: Por hora marcada
5. Detectar pacientes críticos (estabilidade baixa) e priorizar
6. Prescrever medicamentos (pedidos à Farmácia)
7. Solicitar análises laboratoriais
8. Transferir pacientes quando estabilidade atinge 0
9. Atualizar estatísticas

Limites Configuráveis:

- Máximo de emergências simultâneas: 50 (padrão)
- Máximo de consultas simultâneas: 100 (padrão)
- Atendimentos simultâneos: 3

Threads Internas (mínimo 5):

- Thread gestora fila emergências
- Thread gestora fila consultas
- Thread monitorização estabilidade vital
- Threads de atendimento (até 3 simultâneas)

3.3 Blocos Operatórios (Processo Independente)

Função: Gestão de cirurgias em 3 salas especializadas.

Salas Cirúrgicas:

1. **BO1 - Cardiologia**
    - Operações cardíacas complexas
    - Duração: 50-100 time units (configurável)
    - Alta complexidade
2. **BO2 - Ortopedia**
    - Cirurgias ortopédicas
    - Duração: 30-60 time units
    - Complexidade média
3. **BO3 - Neurologia**
    - Neurocirurgias
    - Duração: 60-120 time units
    - Alta complexidade

Recursos Partilhados:

- Equipas médicas: Máximo 2 disponíveis simultaneamente (partilhadas entre salas)
- Cada sala só pode ter 1 cirurgia em curso
- Tempo de limpeza obrigatório entre cirurgias: 10-20 time units

Pré-requisitos Obrigatórios para Cirurgias:

1. Análises pré-operatórias CONCLUÍDAS
2. Medicamentos cirúrgicos DISPONÍVEIS
3. Sala apropriada LIVRE
4. Equipa médica DISPONÍVEL
5. Hora agendada ATINGIDA

Threads Internas (mínimo 3):

- Thread gestora BO1
- Thread gestora BO2
- Thread gestora BO3
- Threads de cirurgias em execução (até 3 simultâneas)

3.4 Farmácia Central (Processo Independente)

Função: Gestão de stock e processamento de pedidos de medicamentos.

Stock de 15 Medicamentos:

Nº	Nome	Stock Inicial	Limiar	Consumo/Pedido
1.	ANALGESICO_A	1000	200	5–15 unidades
2.	ANTIBIOTICO_B	800	150	10–20 unidades
3.	ANESTESICO_C	500	100	20–30 unidades
4.	SEDATIVO_D	600	120	10–15 unidades
5.	ANTIINFLAMATORIO_E	900	180	5–10 unidades
6.	CARDIOVASCULAR_F	400	80	15–25 unidades
7.	NEUROLOGICO_G	300	60	20–30 unidades
8.	ORTOPEDICO_H	700	140	10–20 unidades
9.	HEMOSTATIC_I	350	70	15–20 unidades
10.	ANTICOAGULANTE_J	450	90	10–15 unidades
11.	INSULINA_K	250	50	5–10 unidades
12.	ANALGESICO_FORTE_L	550	110	10–20 unidades
13.	ANTIBIOTICO_FORTE_M	650	130	15–25 unidades
14.	VITAMINA_N	1200	240	5–10 unidades
15.	SUPLEMENTO_O	1000	200	10–15 unidades

Tipos de Pedidos e Prioridades:

1. **URGENT** - Cirurgias (prioridade máxima)
2. **HIGH** - Emergências
3. **NORMAL** - Consultas programadas

Funcionalidades:

- Validar disponibilidade antes de aceitar pedidos
- Reservar medicamentos para pedidos aceites
- Preparação de pedidos: 5-10 time units
- Reposição automática quando stock < limiar
- Quantidade de reposição: configurable (padrão: 2x capacidade)
- Gerar comprovativo de entrega em ficheiro

Threads Internas (mínimo 4):

- Thread processadora pedidos urgentes
- Thread processadora pedidos normais
- Thread monitorização/reposição stock
- Thread geração de estatísticas

3.5 Laboratórios de Análises (Processo Independente)

Função: Execução de exames médicos.

Estrutura - 2 Laboratórios:

LAB1 - Hematológico (análises de sangue):

- HEMO (Hemograma Completo): 10-20 time units
- GLIC (Glicemia): 10-20 time units
- Capacidade: 2 análises simultâneas

LAB2 - Bioquímico:

- COLEST (Colesterol): 15-30 time units
- RENAL (Função Renal): 15-30 time units
- HEPAT (Função Hepática): 15-30 time units
- Capacidade: 2 análises simultâneas

Análise Especial - PREOP (Pré-Operatória):

- Requer LAB1 + LAB2 (sequencial)
- Duração total: 20-40 time units
- Obrigatória para todas as cirurgias

Prioridades:

1. URGENT - Pré-operatórias de cirurgias urgentes
2. NORMAL - Outras análises

Funcionalidades:

- Executar análises em threads dedicadas
- Gerar resultados em ficheiro (formato texto)
- Notificar solicitante via message queue
- Estatísticas de tempo de resposta

Threads Internas (mínimo 2):

- Thread gestora LAB1
- Thread gestora LAB2
- Threads execução análises (até 4 simultâneas: 2 por lab)

---

4. Tipos de Comandos - Especificação Completa

Todos os comandos são recebidos via `input_pipe` e devem seguir rigorosamente o formato especificado.

4.1 COMANDO: Paciente de Emergência

Formato:

```
EMERGENCY {codigo_paciente} init: {tempo_inicial} triage: {1-5} stability:
{100-1000} tests: [{lista_analises}] meds: [{lista_medicamentos}]
```

Exemplos:

```
EMERGENCY PAC001 init: 0 triage: 1 stability: 500 tests: [HEMO,GLIC] meds:
[ANALG_A,ANTIB_B]
EMERGENCY PAC002 init: 10 triage: 3 stability: 800 tests: [] meds:
[ANALG_A]
EMERGENCY PAC003 init: 15 triage: 2 stability: 300 tests: [HEMO] meds: []
```

Parâmetros:

- `codigo_paciente`: Alfanumérico, 5-15 caracteres, único
- `init`: Tempo de criação da thread (≥ 0)
- `triage`: Nível 1 (mais crítico) a 5 (menos crítico)
- `stability`: Unidades de estabilidade vital inicial (100-1000)
- `tests`: Lista de análises (máximo 3): HEMO, GLIC, COLEST, RENAL, HEPAT
- `meds`: Lista de medicamentos (máximo 5): ver lista de 15 medicamentos

Validações Obrigatórias:

- Nível de triagem entre 1 e 5
- Estabilidade ≥ 100
- Análises solicitadas existem
- Medicamentos solicitados existem
- init ≥ tempo simulação atual

Comportamento:

1. Thread criada no tempo `init`
2. Paciente entra na fila de emergências
3. Estabilidade decresce 1 unidade por time unit

4. Se estabilidade ≤ limite crítico (configurável, padrão 50): torna-se CRÍTICO
5. Pedidos de análises e medicamentos enviados conforme necessário
6. Atendimento quando chegar ao topo da fila

4.2 COMANDO: Consulta Agendada

Formato:

```
APPOINTMENT {codigo_paciente} init: {tempo_inicial} scheduled:
{hora_agendada} doctor: {especialidade} tests: [{lista_analises}]
```

Exemplos:

```
APPOINTMENT PAC101 init: 5 scheduled: 100 doctor: CARDIO tests: [COLEST]
APPOINTMENT PAC102 init: 10 scheduled: 150 doctor: ORTHO tests: []
APPOINTMENT PAC103 init: 20 scheduled: 200 doctor: NEURO tests:
[HEMO,GLIC]
```

Parâmetros:

- `codigo_paciente`: Único
- `init`: Tempo de criação
- `scheduled`: Hora marcada para consulta (> init)
- `doctor`: CARDIO | ORTHO | NEURO
- `tests`: Lista de análises (máximo 3)

Validações:

- scheduled > init
- Especialidade válida
- Análises existem

4.3 COMANDO: Cirurgia Programada

Formato:

```
SURGERY {codigo_paciente} init: {tempo_inicial} type: {tipo_cirurgia}
scheduled: {hora_agendada} urgency: {LOW|MEDIUM|HIGH} tests:
[{lista_analises}] meds: [{lista_medicamentos}]
```

Exemplos:

```
SURGERY PAC201 init: 20 type: CARDIO scheduled: 150 urgency: HIGH tests:
[PREOP] meds: [ANEST_C,CARDIOV_F]
SURGERY PAC202 init: 25 type: ORTHO scheduled: 180 urgency: MEDIUM tests:
```

```
[PREOP] meds: [ANEST_C,ORTOPED_H]
SURGERY PAC203 init: 30 type: NEURO scheduled: 200 urgency: LOW tests:
[PREOP] meds: [ANEST_C,NEUROLOG_G]
```

Parâmetros:

- `type`: CARDIO (BO1) | ORTHO (BO2) | NEURO (BO3)
- `urgency`: LOW | MEDIUM | HIGH
- `tests`: **OBRIGATÓRIO incluir PREOP**
- `meds`: Lista de medicamentos cirúrgicos (mínimo 1)

Validações:

- Tipo corresponde a bloco existente
- PREOP está na lista de análises
- Pelo menos 1 medicamento especificado
- scheduled > init

Pré-requisitos Críticos:

- Análise PREOP **DEVE estar concluída** antes de iniciar cirurgia
- Medicamentos **DEVEM estar disponíveis**
- Sala apropriada **DEVE estar livre**
- Equipa médica **DEVE estar disponível**

4.4 COMANDO: Pedido Direto de Medicamentos

Formato:

```
PHARMACY_REQUEST {codigo_pedido} init: {tempo} priority:
{URGENT|HIGH|NORMAL} items:
[{medicamento:quantidade,medicamento:quantidade}]
```

Exemplos:

```
PHARMACY_REQUEST REQ001 init: 5 priority: URGENT items:
[ANALG_A:10,ANTIB_B:5]
PHARMACY_REQUEST REQ002 init: 10 priority: NORMAL items: [VITAMINA_N:20]
```

4.5 COMANDO: Pedido Direto de Análises

Formato:

```
LAB_REQUEST {codigo_pedido} init: {tempo} priority: {URGENT|NORMAL} lab:
{LAB1|LAB2|BOTH} tests: [{lista_analises}]
```

Exemplos:

```
LAB_REQUEST LAB001 init: 8 priority: URGENT lab: LAB1 tests: [HEMO,GLIC]
LAB_REQUEST LAB002 init: 12 priority: NORMAL lab: LAB2 tests:
[COLEST,RENAL]
LAB_REQUEST LAB003 init: 15 priority: URGENT lab: BOTH tests: [PREOP]
```

4.6 COMANDO: Reposição Manual de Stock

Formato:

```
RESTOCK {medicamento} quantity: {quantidade}
```

Exemplo:

```
RESTOCK ANALG_A quantity: 500
```

4.7 COMANDO: Consultar Estado

Formato:

```
STATUS {componente}
```

Componentes: ALL | TRIAGE | SURGERY | PHARMACY | LAB

Exemplo:

```
STATUS ALL
STATUS PHARMACY
```

---

5. Fluxos de Operação Detalhados

5.1 Fluxo: Paciente de Emergência Completo

```
1. COMANDO RECEBIDO
   └─> Validar sintaxe e parâmetros
       └─> Se inválido: Log erro, descartar
       └─> Se válido: Continuar
```

```
2. THREAD CRIADA no tempo init
   └─> Estado inicial: WAITING
   └─> Enviar mensagem ao Centro de Triagem (MQ_NORMAL ou MQ_URGENT se
crítico)

3. CENTRO DE TRIAGEM PROCESSA
   └─> Verificar capacidade (< max_emergencies)
       ├─> Se cheio: Rejeitar paciente, log, atualizar stats
       └─> Se aceita: Inserir na fila ordenada

4. MONITORIZAÇÃO CONTÍNUA DE ESTABILIDADE
   └─> Decrementar 1 unidade por time unit
   └─> Se estabilidade ≤ 0:
       └─> Transferir para outro hospital, log, stats
   └─> Se estabilidade ≤ limite crítico E não era crítico:
       └─> Marcar como CRÍTICO
       └─> Enviar mensagem MQ_URGENT
       └─> Reordenar fila (prioridade máxima)

5. AGUARDAR VEZ NA FILA
   └─> Ordenação: Críticos > Nível triagem > Tempo chegada

6. ANÁLISES (se solicitadas)
   └─> Para cada análise:
       ├─> Enviar pedido aos Laboratórios (MQ)
       ├─> Aguardar conclusão (condition variable ou polling)
       └─> Receber resultados

7. MEDICAMENTOS (se solicitados)
   └─> Enviar pedido à Farmácia (MQ)
   └─> Aguardar disponibilidade
   └─> Receber confirmação de entrega

8. ATENDIMENTO
   └─> Alocar thread de atendimento (máx 3 simultâneas)
   └─> Duração: triage_emergency_duration (config)
   └─> Estado: IN_TREATMENT

9. CONCLUSÃO
   └─> Calcular tempo de espera
   └─> Atualizar estatísticas
   └─> Log conclusão
   └─> Liberar recursos
   └─> Estado: COMPLETED
   └─> Terminar thread
```

5.2 Fluxo: Cirurgia Programada Completo

```
1. COMANDO RECEBIDO E VALIDADO
   └─> Verificar inclusão obrigatória de PREOP
```

```
2. THREAD CRIADA no tempo init
   └─> Estado: PENDING
   └─> Inserir na fila do bloco apropriado (por tipo)

3. SOLICITAR ANÁLISES PRÉ-OPERATÓRIAS (OBRIGATÓRIO)
   └─> Enviar pedido PREOP aos Laboratórios (MQ_URGENT)
   └─> Bloquear até conclusão (pthread_cond_wait ou similar)
   └─> Aguardar notificação de resultados prontos
   └─> Marcar: tests_completed = TRUE

4. SOLICITAR MEDICAMENTOS CIRÚRGICOS
   └─> Enviar pedido à Farmácia (MQ_URGENT)
   └─> Aguardar disponibilidade e preparação
   └─> Marcar: meds_available = TRUE

5. VERIFICAR PRÉ-REQUISITOS
   └─> tests_completed == TRUE
   └─> meds_available == TRUE
   └─> hora atual >= scheduled
   └─> Estado: READY

6. AGUARDAR RECURSOS PARA CIRURGIA
   └─> Sala apropriada livre:
       ├─> CARDIO -> BO1 (sem_wait bo1)
       ├─> ORTHO  -> BO2 (sem_wait bo2)
       └─> NEURO  -> BO3 (sem_wait bo3)
   └─> Equipa médica disponível:
       └─> sem_wait(medical_teams) [contador: max 2]

7. EXECUTAR CIRURGIA
   └─> Marcar sala como OCCUPIED (shm)
   └─> Estado: IN_PROGRESS
   └─> Log início com sala
   └─> Duração: random entre min e max do tipo
   └─> Simulate time passage

8. LIBERAR EQUIPA MÉDICA
   └─> sem_post(medical_teams)
   └─> Log conclusão

9. PERÍODO DE LIMPEZA
   └─> Marcar sala como CLEANING (shm)
   └─> Duração: random(cleanup_min, cleanup_max)
   └─> Simulate time passage

10. LIBERAR SALA
    └─> Marcar sala como FREE (shm)
    └─> sem_post(semáforo_sala)

11. FINALIZAÇÃO
    └─> Calcular tempo total e espera
    └─> Atualizar estatísticas BO
    └─> Estado: COMPLETED
    └─> Terminar thread
```

5.3 Fluxo: Pedido de Farmácia com Stock Esgotado

```
1. PEDIDO RECEBIDO
   └─> Priority: URGENT | HIGH | NORMAL
   └─> Inserir na fila apropriada

2. THREAD PROCESSADORA OBTÉM PEDIDO
   └─> Ordenação: URGENT > HIGH > NORMAL

3. VERIFICAR DISPONIBILIDADE
   └─> Para cada item:
       └─> pthread_mutex_lock(stock_mutex)
       └─> available = current_stock - reserved
       └─> Se available < quantity:
           ├─> STOCK INSUFICIENTE detectado
           ├─> Log warning
           └─> pthread_mutex_unlock
           └─> Ir para passo 4
       └─> pthread_mutex_unlock

4. TRATAMENTO DE STOCK ESGOTADO
   └─> Colocar pedido em fila de espera
   └─> Acordar thread de reposição (pthread_cond_signal)
   └─> Thread fica em wait (pthread_cond_wait) até reposição

5. REPOSIÇÃO AUTOMÁTICA
   └─> Thread monitorização detecta stock < threshold
   └─> Calcular quantidade: (max - current) * multiplier
   └─> Simular tempo de reposição
   └─> pthread_mutex_lock(stock_mutex)
   └─> current_stock += restock_qty
   └─> pthread_mutex_unlock
   └─> Log reposição
   └─> pthread_cond_broadcast (acordar threads em espera)

6. RETOMAR PROCESSAMENTO
   └─> Thread processadora reavalia disponibilidade
   └─> Se agora disponível: continuar
   └─> Senão: voltar a wait

7. RESERVAR MEDICAMENTOS
   └─> Para cada item:
       └─> pthread_mutex_lock
       └─> reserved += quantity
       └─> pthread_mutex_unlock

8. PREPARAR PEDIDO
   └─> Estado: PREPARING
   └─> Simulate preparation time (5-10 ut)
```

```
9. CONFIRMAR ENTREGA
   └─> Para cada item:
       └─> pthread_mutex_lock
       └─> current_stock -= quantity
       └─> reserved -= quantity
       └─> pthread_mutex_unlock

10. FINALIZAÇÃO
    └─> Gerar comprovativo em ficheiro
    └─> Enviar notificação via MQ_RESPONSES
    └─> Atualizar estatísticas
    └─> Estado: DELIVERED
```

5.4 Fluxo: Análise Pré-Operatória (PREOP)

```
1. PEDIDO PREOP RECEBIDO
   └─> Priority: URGENT (sempre)

2. FASE 1: LAB1 (Hemograma)
   └─> Aguardar slot disponível: sem_wait(sem_lab1)
   └─> Log \"PREOP - Fase LAB1 iniciada\"
   └─> Executar análise hematológica
   └─> Duração: lab1_test_duration
   └─> Gerar resultados parciais
   └─> Liberar: sem_post(sem_lab1)

3. FASE 2: LAB2 (Bioquímica)
   └─> Aguardar slot disponível: sem_wait(sem_lab2)
   └─> Log \"PREOP - Fase LAB2 iniciada\"
   └─> Executar análise bioquímica
   └─> Duração: lab2_test_duration
   └─> Gerar resultados parciais
   └─> Liberar: sem_post(sem_lab2)

4. CONSOLIDAÇÃO
   └─> Combinar resultados LAB1 + LAB2
   └─> Gerar relatório completo
   └─> Salvar em ficheiro:
       results/lab_results_{patient_id}_{timestamp}.txt

5. NOTIFICAÇÃO
   └─> Enviar mensagem MQ_RESPONSES
   └─> Tipo: LAB_RESULTS_READY
   └─> Acordar thread de cirurgia em espera
   └─> Log \"Análise PREOP concluída\"

6. ESTATÍSTICAS
   └─> Incrementar total_preop_tests
   └─> Atualizar tempo médio
```

---

6. Comunicação Inter-Processos (IPC) - Especificação Completa

6.1 Message Queues - Implementação

3 Filas de Mensagens:

```
// Chaves para criação
key_t key_urgent = ftok("/tmp", 'U');     // MQ_URGENT
key_t key_normal = ftok("/tmp", 'N');     // MQ_NORMAL
key_t key_resp   = ftok("/tmp", 'R');     // MQ_RESPONSES

// IDs
int mq_urgent;     // Urgent messages (surgeries, critical emergencies)
int mq_normal;     // Normal messages
int mq_responses;  // Responses and notifications
```

Estrutura de Mensagem:

```
typedef struct {
    long msg_priority;              // 1=urgent, 2=high, 3=normal

    // Cabeçalho
    int msg_type;                   // Ver códigos abaixo
    char source[20];                // \"TRIAGE\", \"SURGERY\", etc
    char target[20];                // Componente destino
    char patient_id[15];
    int operation_id;
    time_t timestamp;

    // Dados (formato texto simples ou JSON)
    char data[512];

} hospital_message_t;

// Tipos de mensagem
#define MSG_NEW_EMERGENCY       1
#define MSG_NEW_APPOINTMENT     2
#define MSG_NEW_SURGERY         3
#define MSG_PHARMACY_REQUEST    4
#define MSG_LAB_REQUEST         5
#define MSG_PHARMACY_READY      6
#define MSG_LAB_RESULTS_READY   7
#define MSG_CRITICAL_STATUS     8
#define MSG_TRANSFER_PATIENT    9
#define MSG_REJECT_PATIENT      10
```

Funções Essenciais:

```
int send_message(int mqid, hospital_message_t *msg);
int receive_message_priority(int mqid, hospital_message_t *msg);
int create_all_message_queues();
void cleanup_message_queues();
```

6.2 Shared Memory - 5 Segmentos

SHM1: Estatísticas Globais

```
typedef struct {
    pthread_mutex_t mutex;
    pthread_mutexattr_t mutex_attr;

    // Triagem
    int total_emergency_patients;
    int total_appointments;
    double total_emergency_wait_time;
    double total_appointment_wait_time;
    int completed_emergencies;
    int completed_appointments;
    int critical_transfers;
    int rejected_patients;

    // Blocos Operatórios
    int total_surgeries_bo1;
    int total_surgeries_bo2;
    int total_surgeries_bo3;
    double total_surgery_wait_time;
    int completed_surgeries;
    int cancelled_surgeries;
    double bo1_utilization_time;
    double bo2_utilization_time;
    double bo3_utilization_time;

    // Farmácia
    int total_pharmacy_requests;
    int urgent_requests;
    int normal_requests;
    double total_pharmacy_response_time;
    int stock_depletions;
    int auto_restocks;

    // Laboratórios
    int total_lab_tests_lab1;
    int total_lab_tests_lab2;
    int total_preop_tests;
    double total_lab_turnaround_time;
    int urgent_lab_tests;

    // Globais
```

```
    int total_operations;
    int system_errors;
    time_t system_start_time;
    int simulation_time_units;

} global_statistics_t;
```

SHM2: Estado Blocos Operatórios

```
typedef struct {
    int room_id;
    int status;                     // 0=FREE, 1=OCCUPIED, 2=CLEANING
    char current_patient[15];
    int surgery_start_time;
    int estimated_end_time;
    pthread_mutex_t mutex;
} surgery_room_t;

typedef struct {
    surgery_room_t rooms[3];
    int medical_teams_available;    // 0-2
    pthread_mutex_t teams_mutex;
} surgery_block_shm_t;
```

SHM3: Stock Farmácia

```
typedef struct {
    char name[30];
    int current_stock;
    int reserved;
    int threshold;
    int max_capacity;
    pthread_mutex_t mutex;
} medication_stock_t;

typedef struct {
    medication_stock_t medications[15];
    int total_active_requests;
    pthread_mutex_t global_mutex;
} pharmacy_shm_t;
```

SHM4: Filas Laboratórios

```
typedef struct {
    char request_id[20];
    char patient_id[15];
    int test_type;
```

```
    int priority;
    int status;                     // 0=pending, 1=processing, 2=done
    time_t request_time;
    time_t completion_time;
} lab_request_entry_t;

typedef struct {
    lab_request_entry_t queue_lab1[50];
    lab_request_entry_t queue_lab2[50];
    int lab1_count;
    int lab2_count;
    int lab1_available_slots;       // 0-2
    int lab2_available_slots;       // 0-2
    pthread_mutex_t lab1_mutex;
    pthread_mutex_t lab2_mutex;
} lab_queue_shm_t;
```

SHM5: Log Eventos Críticos

```
typedef struct {
    time_t timestamp;
    char event_type[30];
    char component[20];
    char description[256];
    int severity;                   // 1-5
} critical_event_t;

typedef struct {
    critical_event_t events[1000];
    int event_count;
    int current_index;              // Circular buffer
    pthread_mutex_t mutex;
} critical_log_shm_t;
```

Funções Essenciais:

```
int create_all_shared_memory();
void* attach_shared_memory(key_t key, size_t size);
void initialize_shm_mutexes(void *shm);
void cleanup_shared_memory();
```

6.3 Named Pipes - 5 FIFOs

```
// Criação
mkfifo("input_pipe", 0666);
mkfifo("triage_pipe", 0666);
mkfifo("surgery_pipe", 0666);
```

```
mkfifo("pharmacy_pipe", 0666);
mkfifo("lab_pipe", 0666);

// Abertura
int fd_input = open("input_pipe", O_RDONLY | O_NONBLOCK);
```

6.4 Semáforos POSIX - 7 Semáforos

```
// Criação e inicialização
sem_t *sem_bo1 = sem_open("/sem_surgery_bo1", O_CREAT, 0644, 1);
sem_t *sem_bo2 = sem_open("/sem_surgery_bo2", O_CREAT, 0644, 1);
sem_t *sem_bo3 = sem_open("/sem_surgery_bo3", O_CREAT, 0644, 1);
sem_t *sem_medical_teams = sem_open("/sem_medical_teams", O_CREAT, 0644,
2);
sem_t *sem_lab1 = sem_open("/sem_lab1_equipment", O_CREAT, 0644, 2);
sem_t *sem_lab2 = sem_open("/sem_lab2_equipment", O_CREAT, 0644, 2);
sem_t *sem_pharmacy = sem_open("/sem_pharmacy_access", O_CREAT, 0644, 4);

// Uso
sem_wait(sem_bo1);      // Adquirir
// ... operação crítica ...
sem_post(sem_bo1);      // Liberar

// Cleanup
sem_close(sem_bo1);
sem_unlink("/sem_surgery_bo1");
```

---

7. Sincronização e Prevenção de Problemas

7.1 Requisitos Absolutos (Penalização Grave se Violados)

PROIBIDO - Espera Ativa (Busy Waiting)

```
// MAU - PENALIZAÇÃO GRAVE
while (!condition) {
    // loop vazio ou sleep curto
}

// BOM
pthread_mutex_lock(&mutex);
while (!condition) {
    pthread_cond_wait(&cond_var, &mutex);
}
pthread_mutex_unlock(&mutex);
```

PROIBIDO - Acessos Não Sincronizados

```
// MAU - RACE CONDITION
g_counter++;  // Acesso concorrente sem proteção

// BOM
pthread_mutex_lock(&counter_mutex);
g_counter++;
pthread_mutex_unlock(&counter_mutex);
```

7.2 Estratégias de Sincronização

Mutexes - Proteção de dados partilhados:

```
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);

pthread_mutex_lock(&mutex);
// Secção crítica
pthread_mutex_unlock(&mutex);

pthread_mutex_destroy(&mutex);
```

Variáveis de Condição - Espera eficiente:

```
pthread_cond_t cond;
pthread_cond_init(&cond, NULL);

// Produtor
pthread_mutex_lock(&mutex);
// produzir item
pthread_cond_signal(&cond);
pthread_mutex_unlock(&mutex);

// Consumidor
pthread_mutex_lock(&mutex);
while (queue_empty) {
    pthread_cond_wait(&cond, &mutex);
}
// consumir item
pthread_mutex_unlock(&mutex);
```

Semáforos - Limites de recursos:

```
sem_t sem;
sem_init(&sem, 0, 2);  // Máximo 2

sem_wait(&sem);
```

```
// Usar recurso
sem_post(&sem);

sem_destroy(&sem);
```

7.3 Prevenção de Deadlock

Regras Obrigatórias:

1. **Ordem Consistente de Locks**: Sempre adquirir locks na mesma ordem
2. **Timeouts**: Usar `pthread_mutex_timedlock()` onde apropriado
3. **Try-Lock**: Usar `pthread_mutex_trylock()` e retry
4. **Análise de Dependências**: Verificar grafos de alocação de recursos

Exemplo Correto:

```
// Sempre lock na ordem: mutex_A -> mutex_B
void funcao1() {
    pthread_mutex_lock(&mutex_A);
    pthread_mutex_lock(&mutex_B);
    // ...
    pthread_mutex_unlock(&mutex_B);
    pthread_mutex_unlock(&mutex_A);
}

void funcao2() {
    pthread_mutex_lock(&mutex_A);  // Mesma ordem!
    pthread_mutex_lock(&mutex_B);
    // ...
    pthread_mutex_unlock(&mutex_B);
    pthread_mutex_unlock(&mutex_A);
}
```

---

8. Gestão de Ficheiros

8.1 Ficheiro de Configuração (config.txt)

Formato:

```
# Configurações Globais (comentários permitidos)
TIME_UNIT_MS=500
MAX_EMERGENCY_PATIENTS=50
MAX_APPOINTMENTS=100
MAX_SURGERIES_PENDING=30

# Centro de Triagem
TRIAGE_SIMULTANEOUS_PATIENTS=3
```

```
TRIAGE_CRITICAL_STABILITY=50
TRIAGE_EMERGENCY_DURATION=15
TRIAGE_APPOINTMENT_DURATION=10

# Blocos Operatórios
BO1_MIN_DURATION=50
BO1_MAX_DURATION=100
BO2_MIN_DURATION=30
BO2_MAX_DURATION=60
BO3_MIN_DURATION=60
BO3_MAX_DURATION=120
CLEANUP_MIN_TIME=10
CLEANUP_MAX_TIME=20
MAX_MEDICAL_TEAMS=2

# Farmácia
PHARMACY_PREPARATION_TIME_MIN=5
PHARMACY_PREPARATION_TIME_MAX=10
AUTO_RESTOCK_ENABLED=1
RESTOCK_QUANTITY_MULTIPLIER=2

# Laboratórios
LAB1_TEST_MIN_DURATION=10
LAB1_TEST_MAX_DURATION=20
LAB2_TEST_MIN_DURATION=15
LAB2_TEST_MAX_DURATION=30
MAX_SIMULTANEOUS_TESTS_LAB1=2
MAX_SIMULTANEOUS_TESTS_LAB2=2

# Stock Inicial (nome=stock:threshold)
ANALGESICO_A=1000:200
ANTIBIOTICO_B=800:150
ANESTESICO_C=500:100
# ... (todos os 15 medicamentos)
```

Parsing:

```
typedef struct {
    char key[50];
    char value[100];
} config_param_t;

int load_config(const char *filename, system_config_t *config);
int parse_config_line(char *line, config_param_t *param);
int validate_config(system_config_t *config);
```

8.2 Ficheiro de Log (hospital_log.txt)

Formato de Entrada:

```
[TIMESTAMP] [COMPONENT] [SEVERITY] [EVENT_TYPE] [DETAILS]
```

Níveis de Severidade:

- CRITICAL: Eventos críticos (transferências, falhas graves)
- ERROR: Erros operacionais
- WARNING: Avisos (stock baixo, esperas longas)
- INFO: Eventos informativos normais
- DEBUG: Informação de debug (apenas se #define DEBUG)

Exemplos:

```
[2025-10-23 18:00:05] [SYSTEM] [INFO] [STARTUP] Sistema iniciado - PID:
12345
[2025-10-23 18:00:10] [TRIAGE] [INFO] [COMMAND] Novo comando => EMERGENCY
PAC001
[2025-10-23 18:00:15] [TRIAGE] [ERROR] [VALIDATION] Comando inválido =>
triage: 6
[2025-10-23 18:00:20] [SURGERY] [INFO] [START] PAC003 CIRURGIA CARDIO BO1
iniciada
[2025-10-23 18:00:25] [PHARMACY] [WARNING] [STOCK] ANALG_A stock baixo:
150
[2025-10-23 18:00:30] [LAB] [INFO] [COMPLETE] HEMO PAC001 LAB1 concluída
[2025-10-23 18:00:35] [TRIAGE] [CRITICAL] [EMERGENCY] PAC002 estabilidade
crítica
[2025-10-23 18:00:40] [SURGERY] [INFO] [END] PAC003 CIRURGIA CARDIO BO1
concluída
[2025-10-23 18:00:45] [PHARMACY] [INFO] [RESTOCK] ANALG_A reposto: +1000
[2025-10-23 18:00:50] [TRIAGE] [CRITICAL] [TRANSFER] PAC004 transferido =>
stab=0
```

Implementação:

```c
typedef enum {
    CRITICAL = 1,
    ERROR = 2,
    WARNING = 3,
    INFO = 4,
    DEBUG = 5
} log_severity_t;

void log_event(log_severity_t severity, const char *component,
               const char *event_type, const char *details);

// Deve ser thread-safe (usar mutex)
pthread_mutex_t log_mutex = PTHREAD_MUTEX_INITIALIZER;
```

8.3 Ficheiros de Resultados de Análises

Nome: `results/lab_results_{patient_id}_{timestamp}.txt`

Formato:

```
====================================
RELATÓRIO DE ANÁLISES LABORATORIAIS
====================================
Paciente: PAC001
Data/Hora: 2025-10-23 18:00:30
Laboratório: LAB1

Análises Realizadas:
--------------------

1. HEMOGRAMA COMPLETO
   • Hemoglobina: 14.2 g/dL (normal: 13-17)
   • Leucócitos: 7500/µL (normal: 4000-11000)
   • Plaquetas: 250000/µL (normal: 150000-400000)
   • Hematócrito: 42% (normal: 40-50%)

2. GLICEMIA
   • Glicose em jejum: 95 mg/dL (normal: 70-100)

Observações:
-----------
Todos os parâmetros dentro dos valores de referência normais.
Paciente apto para procedimentos cirúrgicos.

Responsável Técnico: LAB1_TECH_001
Assinatura Digital: ################
====================================
```

8.4 Comprovativos de Farmácia

Nome: `results/pharmacy_delivery_{request_id}_{timestamp}.txt`

Formato:

```
========================================
COMPROVATIVO DE ENTREGA - FARMÁCIA CENTRAL
========================================
Pedido Nº: REQ001
Data/Hora: 2025-10-23 18:00:25
Paciente/Destino: PAC001 (TRIAGE)
Prioridade: URGENT

Medicamentos Fornecidos:
------------------------
```

```
1. ANALGESICO_A
   Quantidade: 10 unidades
   Lote: L2025-001
   Validade: 12/2026

2. ANTIBIOTICO_B
   Quantidade: 5 unidades
   Lote: L2025-045
   Validade: 03/2026

Stock Atual Após Entrega:
-------------------------
- ANALGESICO_A: 990 unidades
- ANTIBIOTICO_B: 795 unidades

Preparado por: PHARM_TECH_002
Conferido por: PHARM_SUPERV_001
Tempo de Preparação: 7 time units
========================================
```

8.5 Snapshots de Estatísticas

Nome: `results/stats_snapshot_{timestamp}.txt`

Gerado: Periodicamente ou via SIGUSR2

Formato (exemplo na secção 10.2)

---

9. Tratamento de Sinais

9.1 SIGINT (Ctrl+C) - Shutdown Gracioso

Handler:

```c
void sigint_handler(int signum) {
    g_shutdown_flag = 1;  // Flag global volatile sig_atomic_t
}
```

Sequência de Shutdown:

```
1. Sinal SIGINT recebido pelo Gestor Central
2. Ativar flag global: g_shutdown = 1
3. Parar de aceitar novos comandos do input_pipe
4. Ler e logar comandos restantes no pipe (sem processar)
5. Aguardar conclusão de operações em curso:
   a. Cirurgias em execução
   b. Análises laboratoriais em processamento
   c. Preparações de medicamentos
```

```
   d. Atendimentos de emergência/consultas
6. Enviar sinal SIGTERM para processos filhos
7. Aguardar terminação de todos os processos (waitpid)
8. Cleanup de recursos IPC:
   a. Fechar e remover named pipes
   b. Remover message queues (msgctl IPC_RMID)
   c. Desanexar e remover shared memory (shmdt, shmctl IPC_RMID)
   d. Fechar e unlink semáforos
9. Gerar relatório final de estatísticas
10. Fechar ficheiro de log
11. Log mensagem de shutdown
12. exit(0)
```

Implementação:

```
void graceful_shutdown() {
    log_event(INFO, "SYSTEM", "SHUTDOWN", "Iniciando encerramento
gracioso");

    // Parar threads
    pthread_cancel(command_reader_thread_id);

    // Aguardar operações (com timeout)
    wait_for_active_operations(60); // 60 segundos max

    // Terminar processos filhos
    kill(pid_triage, SIGTERM);
    kill(pid_surgery, SIGTERM);
    kill(pid_pharmacy, SIGTERM);
    kill(pid_laboratory, SIGTERM);

    waitpid(pid_triage, NULL, 0);
    waitpid(pid_surgery, NULL, 0);
    waitpid(pid_pharmacy, NULL, 0);
    waitpid(pid_laboratory, NULL, 0);

    // Cleanup recursos
    cleanup_message_queues();
    cleanup_shared_memory();
    cleanup_semaphores();
    cleanup_named_pipes();

    // Relatório final
    generate_final_report();

    log_event(INFO, "SYSTEM", "SHUTDOWN", "Encerramento concluído");
    fclose(log_file);

    exit(0);
}
```

9.2 SIGUSR1 - Estatísticas em Tempo Real (Console)

Handler:

```
void sigusr1_handler(int signum) {
    display_statistics_console();
}
```

Comportamento:

- Apenas Gestor Central responde
- Leitura de todos os segmentos SHM
- Apresentação formatada no terminal
- NÃO bloqueia operações em curso
- Usar apenas funções async-signal-safe

Formato de Saída:

```
========================================
ESTATÍSTICAS DO SISTEMA HOSPITALAR
========================================
Timestamp: 2025-10-23 18:30:00
Tempo Operação: 1800 segundos (30 minutos)

CENTRO DE TRIAGEM
-----------------
Total Emergências: 45
Total Consultas: 23
Tempo Médio Espera (Emerg.): 12.5 ut
Tempo Médio Espera (Consul.): 8.3 ut
Pacientes Transferidos: 2
Pacientes Rejeitados: 1
Taxa de Ocupação: 78%

BLOCOS OPERATÓRIOS
------------------
BO1 (Cardiologia):
  Cirurgias: 8 | Tempo Médio: 75.2 ut | Utilização: 85%
BO2 (Ortopedia):
  Cirurgias: 12 | Tempo Médio: 45.6 ut | Utilização: 72%
BO3 (Neurologia):
  Cirurgias: 6 | Tempo Médio: 88.4 ut | Utilização: 68%
Cirurgias Canceladas: 1
Tempo Médio Espera: 23.7 ut

FARMÁCIA CENTRAL
----------------
Total Pedidos: 67
Pedidos Urgentes: 15
Tempo Médio Resposta: 6.8 ut
```

```
Reposições Stock: 3
Esgotamentos: 1

Medicamentos Mais Usados:
  1. ANALGESICO_A: 234 unidades
  2. ANTIBIOTICO_B: 187 unidades
  3. ANESTESICO_C: 156 unidades

LABORATÓRIOS
------------
LAB1: 34 análises | Tempo médio: 14.2 ut | Utilização: 65%
LAB2: 41 análises | Tempo médio: 21.5 ut | Utilização: 71%
Análises Urgentes: 18
Tempo Médio Total: 18.1 ut

GLOBAIS
-------
Total Operações: 186
Throughput: 6.2 ops/min
Erros Sistema: 0
Taxa Sucesso: 98.9%
========================================
```

9.3 SIGUSR2 - Snapshot Estatísticas (Ficheiro)

Handler:

```c
void sigusr2_handler(int signum) {
    save_statistics_snapshot();
}
```

Comportamento:

- Gera ficheiro completo em `results/stats_snapshot_{timestamp}.txt`
- Formato idêntico ao SIGUSR1 mas mais detalhado
- Inclui breakdown por tipo de operação
- Histogramas de tempos de espera
- Não interrompe operações

9.4 SIGCHLD - Monitorização de Processos Filhos

Handler:

```c
void sigchld_handler(int signum) {
    int status;
    pid_t pid;

    while ((pid = waitpid(-1, &status, WNOHANG)) > 0) {
```

```
        if (WIFEXITED(status)) {
            log_event(INFO, "SYSTEM", "CHILD_EXIT",
                      "Processo filho terminou normalmente");
        } else if (WIFSIGNALED(status)) {
            log_event(ERROR, "SYSTEM", "CHILD_KILLED",
                      "Processo filho terminou por sinal");
        }
    }
}
```

---

10. Requisitos de Implementação

10.1 Organização de Código

Estrutura de Diretórios Obrigatória:

```
hospital_system/
├── src/
│   ├── main.c                  # Gestor Central
│   ├── triage.c                # Processo Centro Triagem
│   ├── surgery.c               # Processo Blocos Operatórios
│   ├── pharmacy.c              # Processo Farmácia
│   ├── laboratory.c            # Processo Laboratórios
│   ├── patient_thread.c        # Implementação threads pacientes
│   ├── ipc_utils.c             # Utilitários IPC (MQ, SHM, pipes)
│   ├── sync_utils.c            # Utilitários sincronização
│   ├── log_manager.c           # Gestão de logs
│   ├── stats_manager.c         # Gestão de estatísticas
│   ├── config_parser.c         # Parser de config.txt
│   └── time_simulation.c       # Simulação de tempo
├── include/
│   ├── hospital.h              # Estruturas principais
│   ├── ipc.h                   # Definições IPC
│   ├── sync.h                  # Definições sincronização
│   ├── config.h                # Configurações
│   ├── log.h                   # Interface logging
│   └── stats.h                 # Interface estatísticas
├── config/
│   └── config.txt              # Ficheiro configuração
├── logs/
│   └── hospital_log.txt        # Gerado em runtime
├── results/
│   ├── lab_results/            # Resultados de análises
│   ├── pharmacy_deliveries/    # Comprovativos farmácia
│   └── stats_snapshots/        # Snapshots estatísticas
├── tests/
│   ├── test_basic.sh           # Teste básico
│   ├── test_concurrent.sh      # Teste concorrência
│   ├── test_stress.sh          # Teste stress
│   └── sample_commands/
```

```
│       ├── commands_basic.txt
│       ├── commands_complex.txt
│       └── commands_stress.txt
├── docs/
│   └── README.md               # Instruções uso
├── Makefile
└── REPORT.pdf                  # Relatório técnico
```

10.2 Makefile

```
CC = gcc
CFLAGS = -Wall -Wextra -Werror -pthread -lrt -g -O2
LDFLAGS = -pthread -lrt
INCLUDES = -Iinclude

# Objetos
OBJS = src/main.o src/triage.o src/surgery.o src/pharmacy.o \\
       src/laboratory.o src/patient_thread.o src/ipc_utils.o \\
       src/sync_utils.o src/log_manager.o src/stats_manager.o \\
       src/config_parser.o src/time_simulation.o

# Target principal
TARGET = hospital_system

# Regras
all: $(TARGET)

$(TARGET): $(OBJS)
    $(CC) $(LDFLAGS) -o $@ $^

%.o: %.c
    $(CC) $(CFLAGS) $(INCLUDES) -c -o $@ $<

# Debug build
debug: CFLAGS += -DDEBUG -O0 -ggdb3
debug: clean $(TARGET)

# Cleanup
clean:
    rm -f src/*.o $(TARGET)
    rm -f logs/*.txt
    rm -f results/*/*.txt
    ipcrm -a 2>/dev/null || true

# Testes
test_basic: $(TARGET)
    ./tests/test_basic.sh

test_concurrent: $(TARGET)
    ./tests/test_concurrent.sh
```

```
test_stress: $(TARGET)
    ./tests/test_stress.sh

test: test_basic test_concurrent test_stress

# Verificações
check_memory: $(TARGET)
    valgrind --leak-check=full --show-leak-kinds=all ./$(TARGET)

check_threads: $(TARGET)
    valgrind --tool=helgrind ./$(TARGET)

check_deadlock: $(TARGET)
    valgrind --tool=drd ./$(TARGET)

# Limpeza completa de recursos IPC
ipc_clean:
    @echo \"Removendo recursos IPC...\"
    @ipcs -q | awk '$$3 ~ /^[0-9]/ {print $$2}' | xargs -r ipcrm -q
    @ipcs -m | awk '$$3 ~ /^[0-9]/ {print $$2}' | xargs -r ipcrm -m
    @ipcs -s | awk '$$3 ~ /^[0-9]/ {print $$2}' | xargs -r ipcrm -s
    @rm -f /dev/shm/sem.sem_*
    @rm -f input_pipe triage_pipe surgery_pipe pharmacy_pipe lab_pipe
    @echo \"Recursos IPC removidos.\"

.PHONY: all clean debug test check_memory check_threads check_deadlock
ipc_clean
```

10.3 Código de Debug

```
// No início de cada ficheiro .c
#ifdef DEBUG
    #define DEBUG_PRINT(fmt, ...) \\
        fprintf(stderr, \"[DEBUG][%s:%d:%s] \" fmt \"\
\", \\
                __FILE__, __LINE__, __func__, ##__VA_ARGS__)
#else
    #define DEBUG_PRINT(fmt, ...) do {} while(0)
#endif

// Uso
DEBUG_PRINT(\"Paciente %s entrou na fila\", patient_id);
DEBUG_PRINT(\"Stock ANALG_A: %d unidades\", stock);
```

10.4 Tratamento de Erros - TODOS os System Calls

```
// ERRADO
msgid = msgget(key, IPC_CREAT | 0666);
```

```
// CORRETO
msgid = msgget(key, IPC_CREAT | 0666);
if (msgid == -1) {
    perror(\"msgget failed\");
    log_event(ERROR, \"SYSTEM\", \"MQ_CREATE_FAILED\", strerror(errno));
    cleanup_resources();
    exit(EXIT_FAILURE);
}
```

System calls que DEVEM ter verificação de erro:

- fork, exec
- msgget, msgsnd, msgrcv, msgctl
- shmget, shmat, shmdt, shmctl
- sem_open, sem_wait, sem_post, sem_close, sem_unlink
- open, close, read, write
- pthread_create, pthread_join, pthread_mutex_lock, etc.
- malloc, calloc, realloc

10.5 Gestão de Memória

Regras Obrigatórias:

1. Cada `malloc/calloc` deve ter um `free` correspondente
2. Cada `shmat` deve ter um `shmdt`
3. Verificar sempre retorno de alocações
4. Usar Valgrind para verificar leaks

```
// Padrão RAII
typedef struct {
    void *data;
    size_t size;
} resource_t;

resource_t* create_resource(size_t size) {
    resource_t *res = malloc(sizeof(resource_t));
    if (!res) return NULL;

    res->data = calloc(1, size);
    if (!res->data) {
        free(res);
        return NULL;
    }

    res->size = size;
    return res;
}

void destroy_resource(resource_t *res) {
```

```
    if (res) {
        free(res->data);
        free(res);
    }
}
```

---

11. Testes e Validação

11.1 Casos de Teste Obrigatórios

Teste 1: Operação Básica

```
# tests/commands_basic.txt
EMERGENCY PAC001 init: 0 triage: 3 stability: 500 tests: [HEMO] meds:
[ANALG_A]
APPOINTMENT PAC002 init: 5 scheduled: 50 doctor: CARDIO tests: []
SURGERY PAC003 init: 10 type: ORTHO scheduled: 100 urgency: LOW tests:
[PREOP] meds: [ANEST_C]
```

Teste 2: Prioridade Crítica

```
# Paciente crítico deve ser priorizado
EMERGENCY PAC101 init: 0 triage: 1 stability: 100 tests: [HEMO] meds:
[ANALG_A]
EMERGENCY PAC102 init: 0 triage: 5 stability: 800 tests: [] meds: []
# PAC101 deve ser atendido primeiro mesmo chegando ao mesmo tempo
```

Teste 3: Concorrência Máxima

```
# 10 emergências + 3 cirurgias + múltiplos pedidos
# Verificar: sem deadlocks, sem race conditions, estatísticas corretas
```

Teste 4: Stock Esgotado

```
# Esgotar medicamento
PHARMACY_REQUEST REQ001 init: 0 priority: NORMAL items: [ANALG_A:950]
EMERGENCY PAC201 init: 5 triage: 1 stability: 500 tests: [] meds:
[ANALG_A]
# Deve disparar reposição automática
```

Teste 5: Transferências

```
```

```
# Paciente com estabilidade muito baixa
EMERGENCY PAC301 init: 0 triage: 3 stability: 30 tests: [] meds: []
# Aguardar até estabilidade = 0, verificar transferência
```

Teste 6: Cirurgias Complexas

```
# Todas as cirurgias ao mesmo tempo
SURGERY PAC401 init: 0 type: CARDIO scheduled: 50 urgency: HIGH tests:
[PREOP] meds: [ANEST_C,CARDIOV_F]
SURGERY PAC402 init: 0 type: ORTHO scheduled: 50 urgency: HIGH tests:
[PREOP] meds: [ANEST_C,ORTOPED_H]
SURGERY PAC403 init: 0 type: NEURO scheduled: 50 urgency: HIGH tests:
[PREOP] meds: [ANEST_C,NEUROLOG_G]
# Verificar: análises PREOP concluídas, medicamentos disponíveis, equipas
alocadas
```

Teste 7: Stress Test

```
# 100 comandos aleatórios
# Duração: 1000 time units
# Verificar: sistema não crasha, estatísticas consistentes, sem memory
leaks
```

11.2 Scripts de Teste

test_basic.sh:

```
#!/bin/bash

echo \"=== Teste Básico ===\"
./hospital_system &
PID=$!

sleep 2

cat tests/sample_commands/commands_basic.txt > input_pipe

sleep 10

kill -SIGINT $PID
wait $PID

echo \"Verificando logs...\"
grep -q \"PAC001\" logs/hospital_log.txt && echo \"✓ PAC001 registado\"
grep -q \"CIRURGIA.*PAC003\" logs/hospital_log.txt && echo \"✓ Cirurgia
PAC003 executada\"
```

```
echo \"Teste básico concluído.\"
```

11.3 Verificação com Valgrind

```
# Memory leaks
valgrind --leak-check=full --show-leak-kinds=all \\
         --track-origins=yes --log-file=valgrind_mem.log \\
         ./hospital_system

# Thread errors (race conditions)
valgrind --tool=helgrind --log-file=valgrind_helgrind.log \\
         ./hospital_system

# Deadlocks
valgrind --tool=drd --log-file=valgrind_drd.log \\
         ./hospital_system
```

Critério de Sucesso:

- 0 memory leaks
- 0 race conditions detectadas
- 0 deadlocks possíveis

---

12. Entrega e Avaliação

12.1 Elementos Obrigatórios de Entrega

Ficheiro ZIP com nome: `SO2025_NUMERO_NOME.zip`

Conteúdo:

1. **Código Fonte Completo**
    - Todos os ficheiros .c e .h
    - Makefile funcional
    - Comentários em PT ou EN
2. **Ficheiros de Configuração**
    - config.txt com valores exemplo
    - Scripts de teste (.sh)
    - Ficheiros de comandos de teste
3. **Documentação**
    - README.md com instruções detalhadas
    - REPORT.pdf (relatório técnico, 15-30 páginas)

4. **Evidências de Teste** (opcional mas valorizado)
    - Outputs de testes
    - Logs exemplo
    - Screenshots de execução
    - Relatórios Valgrind

12.2 Estrutura do Relatório (REPORT.pdf)

Obrigatório - 15 a 30 páginas

1. **Capa** (1 página)
    - Título, nome, número, curso, ano letivo
2. **Índice** (1 página)
3. **Introdução** (1-2 páginas)
    - Contextualização
    - Objetivos
    - Organização do documento
4. **Arquitetura do Sistema** (4-6 páginas)
    - Diagrama geral
    - Descrição de cada processo
    - Diagrama de threads
    - Fluxos de comunicação
    - Justificação de decisões arquiteturais
5. **Mecanismos IPC** (3-4 páginas)
    - Message Queues (design e uso)
    - Shared Memory (estruturas e acesso)
    - Named Pipes (utilização)
    - Semáforos (estratégia)
6. **Sincronização** (3-5 páginas)
    - Identificação de secções críticas
    - Estratégias de sincronização
    - Prevenção de deadlock
    - Prevenção de race conditions
    - Gestão de prioridades
7. **Algoritmos de Escalonamento** (2-3 páginas)
    - Triagem de pacientes
    - Escalonamento de cirurgias
    - Priorização farmácia
    - Gestão laboratórios

- Análise de eficiência

8. **Implementação** (4-6 páginas)
    - Decisões técnicas importantes
    - Estruturas de dados
    - Snippets de código crítico (com explicação)
    - Gestão de erros
    - Otimizações
9. **Testes e Validação** (2-3 páginas)
    - Estratégia de testes
    - Casos de teste
    - Resultados
    - Análise de performance
    - Verificações Valgrind
10. **Estatísticas e Análise** (2-3 páginas)
    - Métricas recolhidas
    - Throughput
    - Tempos de espera
    - Utilização de recursos
    - Gráficos
11. **Análise Crítica** (2-3 páginas)
    - Dificuldades encontradas
    - Limitações
    - Melhorias possíveis
    - Lições aprendidas
12. **Conclusão** (1-2 páginas)
    - Síntese
    - Objetivos alcançados
    - Trabalho futuro
13. **Bibliografia**
    - Manuais Linux
    - Documentação POSIX
    - Livros referência
14. **Anexos** (opcional)
    - Código completo
    - Outputs extensos
    - Diagramas adicionais