# **Informe Técnico: Simulación de Asedio de Castillo**

**Comparación de Mecanismos de Sincronización en Sistemas Concurrentes**

**Resumen:**

El presente informe analiza la implementación de una simulación de asedio de castillo desarrollada en lenguaje C basado en el videojuego de Paradox Studios Europa Universalis IV, comparando la eficiencia de tres mecanismos de sincronización: semáforos, mutex y espera activa.

**1. Introducción**

**1.1 Descripción del Problema**

La simulación implementa un sistema concurrente que modela el asedio de un castillo medieval en un contexto de videojuego. El problema presenta las siguientes características:

* **Atacantes**: 4 grupos de 3,000 unidades de infantería cada uno (12,000 total)
* **Defensores**: Guarnición de 6,000 unidades
* **Objetivo**: Comparar la eficiencia de tres mecanismos de sincronización diferentes

**1.2 Relevancia en Sincronización**

Este problema es representativo de sistemas concurrentes donde múltiples procesos (hilos) deben acceder a recursos compartidos de manera coordinada. La elección del mecanismo de sincronización puede impactar significativamente el rendimiento del sistema.

**1.3 Objetivo del Estudio**

Evaluar y comparar la eficiencia temporal de tres mecanismos de sincronización (semáforos, mutex y espera activa) en un escenario de alta concurrencia, midiendo latencia promedio y número de rondas completadas.

**1.4 Repositorio**

**Enlace del repositorio:** [**Asedio de la fortaleza**](https://github.com/StivenMelendez/threads-en-C/blob/master/threads-taller.c)

**1.5 Prueba**

* ***git clone*** [***https://github.com/StivenMelendez/threads-en-C.git***](https://github.com/StivenMelendez/threads-en-C.git)
* ***gcc threads-taller.c -o ejecutable***
* ***./ejecutable***

**2. Contexto**

**2.1 Mecanismos de Sincronización Implementados**

**2.1.1 Semáforos**

* **Implementación**: *sem\_t asediando* inicializado con valor 1
* **Funciones**: *sem\_wait() y sem\_post()*
* **Características**: Mecanismo de sincronización de alto nivel con bloqueo eficiente

**2.1.2 Mutex**

* **Implementación**: *pthread\_mutex\_t mutex\_asedio*
* **Funciones**: *pthread\_mutex\_lock() y pthread\_mutex\_unlock()*
* **Características**: Exclusión mutua específica para un solo hilo a la vez

**2.1.3 Espera Activa**

* **Implementación**: Variables globales *turno y bandera\_ocupado*
* **Mecanismo**: Sondeo continuo hasta obtener acceso a la sección crítica
* **Características**: Consumo activo de CPU durante la espera

**2.1.4 Descripción de variables globales**

* int ASEDIO; // LLEVA EL PORCENTAJE DE ASEDIO
* int ATACANTES[GRUPOS\_DE\_INFANTERIA]; //GRUPOS DE ATAQUE
* int DEFENSORES; //GUARNICION DEFENSIVA
* int RESERVAS; //MANPOWER DEL ATACANTE
* int RONDAS; //RONDAS DE ASEDIO “DIARIAS”
* int id[GRUPOS\_DE\_INFANTERIA]; //PARA GUARDAR EL GRUPO QUE ATACA
* int estado\_castillo;// CASTILLO CAIDO = 0, CASTILLO EN PIE = 1
* int enfermedad; // ENFERMEDAD EN EL FUERTE. SI = 0, NO = 1
* int turno; //PARA CONTROLAR LA ESPERA ACTIVA Y HACER SONDEO
* int bandera\_ocupado;//BANDERA PARA HACER ESPERA ACTIVA
* int espera\_apagada;//PARA CONTROLAR LA ESPERA
* int rmu, rse, rea;//PARA GUARDAR LAS RONDAS DE CADA METODO
* pthread\_t HILOS[GRUPOS\_DE\_INFANTERIA];//HILOS
* sem\_t asediando;//SEMAFORO PARA EL ASEDIO
* pthread\_mutex\_t mutex\_asedio;//MUTEX PARA EL ASEDIO
* /\*ESTRUCTURA PARA ALMACENAR LOS DATOS DE LA MEDICION\*/
* typedef struct{long inicio, fin, acum; double prom;}medicion;
* medicion
* para\_mutex = {0, 0, 0, 0},
* para\_semaforo = {0, 0, 0, 0},
* para\_espera\_activa = {0, 0, 0, 0};
* /\*PARA ALMACENAR LOS MEJORES TIEMPOS\*/
* long mejor\_tiempo\_semaforo;
* long mejor\_tiempo\_mutex;
* long mejor\_tiempo\_espera\_activa;

**2.1.5 Descripción de funciones**

* int solucion\_semaforos(); //ejecutor para semaforos
* int solucion\_mutex(); //ejecutor para mutex
* int solucion\_espera\_activa(); //ejecutor para espera activa
* void inicializacion(); //para reiniciar variables globales
* int calculador\_de\_probabilidad(int *probabilidad*);//para calcular las probabilidades
* void\* asedio\_con\_semaforos(void\* *arg*);//asedio con semaforo
* void\* asedio\_con\_mutex(void\* *arg*);//asedio con mutex
* void\* asedio\_con\_espera\_activa(void\* *arg*);//asedio con espera activa
* int cantidad\_infanteria();//calcula la infantería restante
* void finalizador();//función estetica
* void comparar\_metodos();//compara los metodos
* long para\_medir\_tiempo(long *in*, long *fi*);//mide el tiempo
* long para\_hacer\_promedio(long *acum*, int *ro*);//para hacer un promedio

**2.2 Arquitectura del Sistema**

El sistema utiliza 4 hilos concurrentes (uno por grupo de infantería) que compiten por acceso a una sección crítica donde se actualizan las variables compartidas del estado del asedio.

**3. Metodología**

**3.1 Variables Independientes**

* **Número de hilos**: 4 hilos concurrentes (constante)
* **Mecanismo de sincronización**: Semáforos, Mutex, Espera Activa
* **Carga de trabajo**: Cálculos probabilísticos y actualización de estado

**3.2 Variables Dependientes**

* **Latencia promedio**: Tiempo en nanosegundos por ronda de asedio
* **Throughput**: Número total de rondas completadas
* **Tiempo total de ejecución**: Duración de cada método de sincronización

**3.3 Controles y Factores de Confusión**

* **Inicialización consistente**: Mismo estado inicial para todas las pruebas
* **Semilla aleatoria**: *srand(time(NULL))* para variabilidad controlada
* **Condiciones de terminación**: Criterios uniformes para finalizar simulaciones

**3.4 Procedimiento de Medición**

*struct timespec tiempo\_inicio, tiempo\_fin;*

*clock\_gettime(CLOCK\_MONOTONIC, &tiempo\_inicio);*

*// Sección crítica*

*clock\_gettime(CLOCK\_MONOTONIC, &tiempo\_fin);*

**Herramientas de medición**:

* *clock\_gettime()* con resolución de nanosegundos
* Acumulación de tiempos por ronda
* Cálculo de promedio: *tiempo\_total / número\_rondas*

**4. Implementación Técnica**

**4.1 Arquitectura de Hilos**

Cada hilo ejecuta la función correspondiente al mecanismo de sincronización:

* asedio\_con\_semaforos()
* asedio\_con\_mutex()
* asedio\_con\_espera\_activa()

**4.2 Sección Crítica**

La sección crítica incluye:

* Actualización del estado del asedio (progreso 0-100%)
* Cálculo de bajas para atacantes y defensores
* Gestión de reservas y reposición de tropas
* Verificación de condiciones de victoria/derrota

**4.3 Modelo Probabilístico**

El sistema implementa un modelo estocástico con las siguientes probabilidades:

**Atacantes**:

* 95% probabilidad de 5% bajas por atrición
* 50% probabilidad de 10% bajas adicionales por asedio
* 10% probabilidad de 10% bajas por desorganización

**Defensores**:

* 5% probabilidad de 10% bajas por atrición
* 10% probabilidad de 10% bajas por combate directo
* Posibilidad de enfermedad cada 30 rondas (50% probabilidad)

**5. Resultados**

**5.1 Métricas de Rendimiento**

* printf("\n\n\*\*\*\*\* COMPARACION DE RENDIMIENTO \*\*\*\*\*\n");
* printf("Metodo         || Mejor Tiempo (ms)      || RONDAS\n");
* printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*||\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|| \n");
* printf("SEMAFOROS      || [%.5f]\*\*\*\*\*\*\*\*\*\*\*\*\*\*|| [ %d ]\n", mejor\_tiempo\_semaforo / 1000000.0, rse);
* printf("MUTEX          || [%.5f]\*\*\*\*\*\*\*\*\*\*\*\*\*\*|| [ %d ]\n", mejor\_tiempo\_mutex / 1000000.0, rmu);
* printf("ESPERA ACTIVA  || [%.5f]\*\*\*\*\*\*\*\*\*\*\*\*\*\*|| [ %d ]\n", mejor\_tiempo\_espera\_activa / 1000000.0, rea);
* printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");
* // Determinar el método más rápido basado en el mejor tiempo
* if (mejor\_tiempo\_semaforo <= mejor\_tiempo\_mutex && mejor\_tiempo\_semaforo <= mejor\_tiempo\_espera\_activa) {
* printf("El metodo mas rapido es: SEMAFOROS\n");
* }
* else if (mejor\_tiempo\_mutex <= mejor\_tiempo\_semaforo && mejor\_tiempo\_mutex <= mejor\_tiempo\_espera\_activa) {
* printf("El metodo mas rapido es: MUTEX\n");
* }
* else {
* printf("El metodo mas rapido es: ESPERA ACTIVA\n");
* }

**5.2 Análisis Comparativo**

**Hipótesis de rendimiento**:

1. **Semáforos**: Eficiencia alta por optimización del kernel
2. **Mutex**: Rendimiento similar a semáforos, posible ligera ventaja
3. **Espera Activa**: Menor eficiencia por consumo continuo de CPU

**5.3 Factores de Variabilidad**

* Carga del sistema durante la ejecución
* Variabilidad aleatoria del modelo probabilístico
* Componentes del equipo
* Entorno de pruebas

**6. Limitaciones y Consideraciones**

**6.1 Limitaciones del Diseño**

* **Asedio simplificado**: El modelo no considera todos los factores de un asedio real.
* **Medición desigual**: En la gran mayoría de casos los asedios son desiguales.

**6.2 Consideraciones de Escalabilidad**

* **Número fijo de hilos**: Limitado a 4 grupos de infantería
* **Recursos compartidos**: Un solo recurso crítico puede crear cuello de botella
* **Sistema de aleatoriedad (Hardware-based):** Implementar un sistema de aleatoriedad basado en hardware que use el movimiento del ratón, valores ambientales etc.
* **Implementar múltiples procesos:** para poder simular y hacer mejores pruebas se tendría que implementar muchos asedios constantes para poder medir la efectividad a mayor escala.

**7. Conclusiones**

**7.1 Aplicabilidad Práctica**

Este estudio proporciona resultados valiosos para:

* **Desarrollo de videojuegos**: Selección de mecanismos de sincronización eficientes para problemas de productor-consumidor.
* **Optimización de rendimiento**: Comparación cuantitativa de alternativas

**7.2 Rendimiento:**

1. Estas pruebas demuestran que para experimentos pequeños la espera activa da mejores resultados.
2. Aunque en nuestra hipótesis los semáforos dejarían de los mejores resultados, no logro mejores resultados que los otros métodos
3. Mutex es inestable en las pruebas ya que presenta variabilidad con valores extraños en comparación con espera activa.