-(WHAT) IN COSA CONSISTE:

Il progetto consiste nell'implementazione dei semafori e delle funzioni che li regolano nel sistema operativo disastrOS.

Un semaforo è un tipo di dato astratto che permette a più processi di accedere alla sezione critica di un programma o ad una risorsa condivisa (nel nostro caso operazione di lettura/scrittura su un buffer circolare) in mutua esclusione e in modo sincronizzato.

Un semaforo consiste in una variabile intera e sfrutta due operazioni per gestire l'accesso a sezione critica:

1) WAIT: quando un processo vuole accedere alla sezione critica o richiedere accesso alla risorsa viene eseguita una wait sul semaforo e il suo contatore viene decrementato;

2)POST: quando un processo abbandona la sezione critica o rilascia la risorsa viene eseguita una post sul semaforo e il suo contatore viene incrementato.

-(HOW) IMPLEMENTAZIONE:

Sono state implementate all'interno del codice sorgente del sistema quattro system call che gestiscono il funzionamento dei semafori:

1)semOpen: crea un semaforo con un id e un contatore scelti dall'utente.

Se il semaforo in questione è già presente nella lista globale dei semafori apre solamente un nuovo descrittore (che viene inserito anche come entry della lista dei descrittori del processo corrente), altrimenti lo alloca e lo aggiunge alla lista dei semafori.

La semOpen inoltre crea un puntatore al descrittore, che serve da entry nella lista dei descrittori attivi del semaforo.

Infine, incrementa il contatore dei descrittori attivi del processo in esecuzione e se ha funzionato tutto correttamente imposta come valore di ritorno il file descriptor del semaforo appena aperto (altrimenti il valore di ritorno dipende dal tipo di errore).

2)semClose: dato in input il file descriptor di un semaforo lo rimuove e lo dealloca dalla lista dei descrittori del processo in esecuzione.

Rimuove e dealloca anche il puntatore al descriptor nella lista dei descrittori attivi del semaforo, e se il semaforo non ha più descrittori attivi lo rimuove dalla lista globale e lo dealloca.

Infine, decrementa il contatore dei descrittori attivi e se ha funzionato tutto correttamente imposta il valore di ritorno a zero (altrimenti il valore di ritorno dipende dal tipo di errore).

3)semWait: chiamata a sistema che, dato in input un file descriptor, decrementa il valore del semaforo associato. Tale processo viene iterato ad ogni chiamata, finché, talora dovesse accadere, il contatore diventa negativo. In questo caso la semWait sposta il descrittore, e il rpocesso ad invocante la funzione, nelle rispettive liste d'attesa. Infine, messo in "Waiting" il processo, pone in esecuzione un nuovo processo. Nello specifico, verrà preso il primo della lista dei processi nello stati di "Ready", eseguendo, sostanzialmente, un Context Switch. Se tutto sarà andato a buon fine, imposta il valore di ritorno pari a 0, altrimenti sarà indicato dal tipo di errore avvenuto.

4)semPost: chiamata a sistema usata per incrementare il contatore di un semaforo associato al file descriptor preso in Input. Qualora tale contatore dovesse essere nullo o negativo dopo l'incremento, il descrittore e il processo invocante la sys call verranno spostati nelle rispettive liste Ready Queue, impostando lo stato del processo attuale in Ready. Qualora fosse funzionato tutto correttamente, la semWait imposterebbe il valore di ritorno a 0, o pari al tipo di errore avvenuto.

Oltre alle chiamate a sistema (Sys Call), è stato implementato un programma di test, usando lo scheletro del programma esistente, per creare un esempio di funzionamento dei semafori. Come contesto è stato preso il modello dei N-produttori/N-consumatori, ponendo attenzione anche alla realizzazione dei mutex per evitare Race condition. É stata scritta una funzione "child" che si occupa di gestire la scrittura e lettura

da Buffer, mentre la funzione Init si occupa di gestire la creazione e il lancio dei Thread associati. In questo caso è stato scelto un Buffer da 100 elementi, inizializzati a 0, che subisce operazioni di lettura e scrittura da più processi in modo sincronizzato, ciò è stato possibile proprio grazie all'utilizzo dei semafori/Mutex e delle funzioni appena descritte.

-(HOW TO RUN) LANCIO DEL PROGRAMMA:

Per eseguire il Test del programma bisogna andare nella cartella "codice_sorgente", aprire il terminale nella cartella, compilare tutti i file con"make" ed infine lanciare l'eseguibile "./disastrOS_test".

ANTONIO MARIGLIANO - 1745765
DAVIDE MASSIMEI - 1770456