Progetto SO 2024/25

Ufficio delle Poste – Versione completa

Arnaud Ismael Wenyam – 1113980 - ismael.arnaud@edu.unito.it

Introduzione

Il presente progetto, simula il funzionamento di un ufficio postale. A tal fine sono presenti i seguenti processi: ***direttore, erogatore_ticket, sportello, operatore, utente***.

Il processo direttore è deputato alla creazione dei processi qui sopra e alla creazione delle risorse ipc, necessarie alla simulazione.

Tutti i parametri di configurazione sono letti a runtime da un file di configurazione denominato, ***simulation_configuration.conf,*** permettendo la modifica di quest'ultimi senza la necessita di nuova compilazione dei sorgenti.

Inoltre è presente un ulteriore eseguibile *add_user* che una volta eseguito comunica a runtime al *direttore* l'aggiunta di ***n_new_users*** alla simulazione.

La simulazione termina grazie a due file di configurazione: ***config_timeout.conf*** e ***config_explode.conf***.

Nel caso di ***config_timeout.conf,*** la simulazione termina quando il valore ***timeout*** raggiunge o eccede il valore ***sim_duration*** definito in ***simulation_configuration.conf*** *mentre* nel caso di ***config_explode.conf***, la simulazione termina quando il valore di ***explode*** raggiunge o eccede il valore di ***explode_threshold*** in ***simulation_configuration.conf***.

La simulazione produce delle statistiche giornaliere e totali stampate dal direttore al termine di ogni giornata.

Tutte le statistiche prodotte vengono anche salvate in files csv che sono:

daily_stats.csv, extra_daily_stats.csv, operator_ratio.csv, total_stats.csv, extra_total_stats.csv.

Nel file README.md del progetto sono presenti le istruzioni su come eseguire la simulazione.

Processi

Direttore

Il processo direttore si occupa della creazione delle risorse IPC, all'interno del processo vengono usati semafori, memoria condivisa e code di messagi.

Le chiavi per la creazione delle risorse sono reperibile all'interno dei rispettivi header files all'interno della directory ***include/***.

Il direttore si occupa anche della creazione delle risorse utili alla simulazione, usando la system call ***execve***, dato che quest'ultima sostituisce l'immagine del processo chiamante con quella del processo chiamato, il direttore esegue una fork per ogni processo da creare con execve, in modo da mantenere la propria immagine.
E questa è la formula usata per la creazione di tutti i processi partecipanti alla simulazione.

nofProcess → [fork → execve("./eseguibile") × nofProcess]

Terminata la creazione dei processi, il processo direttore attende la richiesta di ruoli da parte dei processi appartenenti a *operatore_group* tramite ***dirMsgQueue***. Ad ogni operatore assegna un ruolo garantendo che per ogni servizio vi sia almeno un operatore che lo esegua, una volta questo requisito garantito, assegna casualmente un ruolo agli operatori restanti.

Successivamente attende il completamento dell'inizializzazione dei processi appartenenti ai gruppi ***operatore_group, utente_group, erogatore_group***, prima dell'avvio della simulazione che equivale alla prima giornata della simulazione.

Questo processo di sincronizzazione viene eseguito grazie all'uso di semafori.

All'avvio di ogni giornata il direttore decide in modo casuale quale servizio verra fornito dall'ufficio, garantendo la disponibilità di almeno un servizio.

Successivamente riceve le richieste di ruolo da parte dei processi appartenenti a ***sportello_group,*** tramite ***dirMsgQueue,*** e ad ogni sportello assegna in modo casuale la funzione da svolgere per quella giornata ed aggiorna la memoria condivisa riguardanti I sportelli e inizializza la memoria condivisa riguardante le statistiche dei sportelli.

In particolare la memoria condivisa per le statitische dei sportelli serve per il ratio tra operatori e sportello.

La gestione di ogni giornata, in particolare la sincronizzazione tra i processi ed il direttore avviene tramite semafori.

Al termine della giornata lavorativa il direttore, accede al file **config_timeout.conf** in mutua esclusione incrementando il valore di timeout, comunica ai vari processi ***eod*** (end of day) questo viene fatto tramite rilascio di semafori sui quali i vari processi erano in attesa.

Poi esegue la stampa a terminale delle statistiche, il loro dumping nei rispettivi files e l'inizializzazione delle statistiche giornaliere.

Infine prima di passare alla giornata successivamente esegue una msgrcv, per controllare se vi sia una richiesta di aggiunta di processi utente, in caso positivo l'aggiunta degli utenti viene eseguita, altrimenti controlla in mutua esclusione che il valore di ***explode*** in ***config_explode.conf*** non abbia eguagliato o superato il valore di ***explode_threshold.***

Al termine della simulazione il direttore esegue il dumping delle statistiche totali, della rimozione delle risorse ipc, della stampa della causa della terminazione della simulazione, e infine termina.

Erogatore

In prima istanza l'erogatore esegue la fase di inizializzazione che comprende il recuparo delle risorse ipc, come i semafori per la sincronizzazione, la coda di messaggio **ticketsMsgQueue** attraverso cui ricevere le richieste di tickets da parte degli utenti e attraverso cui le fornire i tikcets richiesti e il recupero delle memorie condivise sia dei servizi che dei sportelli.

La memoria condivisa dei servizi è utile a vedere quali servizi sono abilitati per il giorno i mentre la memoria condivisa dei sportelli è utile a vedere quali sono i sportelli che forniscono il servizio nella richiesta del ticket.

Al termine della fase di inizializzazione viene eseguito una fork.

Questo permette di dividere il carico di lavoro tra il parent ed il child.

Il parent si occupa della sincronizzazione con il direttore mentre il child si occupa della gestione di richieste di tickets.

All'avento di un cambio di stato come, ***eod*** (end of day), ***eos*** (end of simulation), ***sod*** (start of day), il child viene notificato dal parent.

Una volta ricevuta la richiesta di ticket dall'utente, l'erogatore (child) costruisce il ticket accedendo in mutua esclusione alla memoria condivisa dei servizi e dei sportelli, inoltre l'operatore genera in maniera casuale un tempo di erogazione del servizio che è all'intorno +-50% del tempo medio riscontrato per il servizio.

Qualora l'erogatore (child) riceva una richiesta di ticket mentra lo stato della simulazione è in eod, invierà all'utente un ticket notificando all'utente che la giornata è terminata.

Al termine dell'inoltro del ticket, l'erogatore (child) attende la seguente richiesta di ticket da parte di un utente o una notifica dal parent.

Come detto poco sopra il parent si occupa di sincronizzarsi con il direttore attraverso dei semafori, inoltre all' ***eod***, il parent controlla se ***eos*** per ***timeout*** o ***explodeThreshold***, qualora positivo notifica al child il cambio di stato, e attende la sua terminazione prima di terminare anch'egli.

Sportello

Il processo sportello al termine della sua inizializzazione, che comporta il recupero delle risorse ipc, come semofari e memoria condivisa, in particolare la memoria condivisa contenente i sportelli, richiede il ruola per la giornata tramite la coda di messaggi, ***dirMsgQueue***.

Ricevuto il ruolo per la giornata , accede alla memoria condivisa contenente I sportelli, ed aggiorna i dati dello sportello che esegua la sua mansione.

Aggiorna i campi quali: ***deskAvailable*** *(informazione utile all'operatore)*, ***deskSemId*** *(semaforo privato creato dallo sportello, sul quale l'operatore dovra reserve_sem per poter lavorare)* e ***deskSemun*** valorizzato di default a 0;

Terminato questa parte lo sportello attenda la fine della giornata su di un primo semoforo che una volta conquistato gli permetterà di inizializzare lo sportello di sua pertinenza e successivamente controllare se la simulazione è terminata per ***timeout*** o ***explodeThreshold***, in caso contrario, ricomincia il ciclo chiedendo un ruolo per la giornata successiva al direttore.

Se la simulazione termina lo sportello procede alla cancellazione della risorsa ipc privata cioè il semaforo, prima di terminare.

Operatore

Il processo operatore terminata la fase di inizializzazione che comprende il recupero delle risorse ipc, quali semaforo per la sincronizzazione, la memoria condivisa dei servizi, delle statistiche, dei sportelli e delle statististiche dei sportelli per il calcolo del rapporto tra operatori e sportelli, riceve anche il proprio ruolo per l'intera durata della simulazione, solo allora la fase inizializzazione dell'operatore termina, quando egli esegue una fork in quanto il suo carico di lavoro come nel caso dell'erogatore sarà suddiviso tra il parent ed il child.

Il child si occuperà della gestione delle richieste da parte dell'utente mentre il padre si occuperà della gestione della sincronizzazione con il direttore.

Quindi terminata la fase di inzializzazione il parent attende l'inizio della simulazione/giornata lo comunica al child, tramite la ***serviceMsgQueue***, il quale compete con gli altri operatore per trovare un sportello libero da occupare.

Come detto poc'anzi la funzione del padre è la sincronizzazione con il direttore, la comunicazione al figlio dei cambiamenti di stato nella simulazione ***eod***, ***eos***, ***sod***, e della rimozione della risorse ipc privata in creato dell'operatore (semaforo) su cui l'utente potra mettersi in fila (reserve_sem).

Le mansioni demandate al child sono l'occupazione di uno sportello (reserve_sem), l'attesa che si liberi lo sportello per la pausa di un altro operatore se lo sportello non è disponibile.

E la ricezione del richieste di servizio dall'utente e l'esecuzione di questi servizi.

Un altra funzione importante dell'operatore è l'aggiornamente delle statistiche.

Utente

Il processo utente, terminata la fase di inizializzazione che comporta il recupero delle risorse ipc, come semafori per la sincronizzazione con il direttore, code di messaggi *(**ticketsMsgQueue** e **serviceMsgQueue**)* e delle memorie condivise contente i servizi attende dal direttore il segnale per l'inizio della simulazione e quindi della giornata.

In prima istanza l'utente decide casualmete secondo un valore compreso tra ***[p_serv_min, p_serv_max]*** se andare in ufficio postale o meno.

Successivamente sceglie casualmente la quantita ***(q)*** di servizi di cui ha bisogno nella giornata i con ***1 <= q <= n_requests*.**

La scelta del ***servizio[0]***,..,***servizio[q-1]*** avviene anch'essa in modo casuale e la richiesta del ticket per il servizio avviene dopo un controllo della disponibilità del servizio nella giornata i attraverso la shm dei servizi.

Successivamente avviene la richiesta del ticket per il ***servizio[q]*** da parte dell'utente all'erogatore_ticket, la richiesta del ticket per il ***servzio[i]*** avviene solamente dopo la conclusione dell'espletamento del ***servizio[i-1]*** da parte dell'operatore.

Ricevuto il ticket, l'utente se non ***eod***, si prenota (reserve_sem) allo sportello, che è un semaforo privato creato dall'operatore.

Quando il suo turno giunge l'utente inoltra la richiesta di servizio all'operatore, tramite la ***serviceMsgQueue*** e attende una risposta da parte dell'operatore che può essere:

- ok → servizio completato con successo

- eod → end of day

Nel caso di eod, l'utente incrementa il valore di ***explode*** in ***config_explode.conf*** e rinuncia ai servizi rimasti se ***n_requests > 1*** e controlla se ***eos*** per ***explode*** o ***timeout***, prima di attendere la giornata successiva.