**Progetto Sistemi Operativi 2022/2023**

**Collaboratori:**

* Monaldi Stefano (866430)
* Parigi Enrico (948760)
* Schiavone Marco (976640)

**Progetto:** Simulazione Navi e porti – Versione da 24

**Indice:**

1. Come avviare la simulazione
2. Scelte implementative
3. Spiegazione del progetto
4. Pro
5. Contro
6. **Come avviare la simulazione**

Per avviare la simulazione abbiamo fatto uso della *make utility* per poter compilare e avviare nel seguente modo:

* Il file ***config.txt*** contiene, linea per linea, i parametri da passare alla simulazione. Qualora si volesse aggiungere una configurazione di parametri specifica, basterà inserirli nel file su una linea e usare come parametro di chiamata il numero della linea precedente (ad esempio, se scrivo una nuova configurazione a riga 20, dovrò lanciare il programma con parametro 19).
* Una volta decisa la configurazione con cui voler lanciare la simulazione, basterà scrivere a terminale: make run var=X

Dove X è il numero della configurazione scelta, definito come sopra.

* Le configurazioni presenti nella consegna del progetto come esempi sono state inserite come prime 4 configurazioni; per avviare la simulazione con tali parametri basterà usare rispettivamente i valori da 1 a 4.
* Abbiamo inoltre aggiunto la possibilità di eseguire la fase di stampa del dump in mutua esclusione. A tal fine basterà scrivere:

make runME var=X

1. **Scelte implementative**

* Abbiamo utilizzato delle risorse IPC (System V). Più precisamente ci sono 4 **shared memory**, 4 **semafori** e una **coda di messaggi**.
* Si fa uso dei segnali **SIGUSR1** (lancio stato di dump), **SIGUSR2** (fine simulazione) e **SIGALRM** (cadenza di lancio dello stato di dump) per la comunicazione tra i processi.

MEMORIE CONDIVISE:

L’uso delle shared memory permette una maggiore velocità delle operazioni rispetto alle pipe, alle FIFO e alle code, in quanto limita le system call a un numero ridotto, diminuendo spesso i tempi di esecuzione dei blocchi. Di contro va gestita con cautela, motivo per cui esistono i 2 semafori (mercato e dump).

Le memorie condivise sono definite come (void \*), ma organizzate e utilizzate secondo le strutture definite in *definitions.h*.

SEMAFORI:

* *banchine* è un set dato per scontato, come da consegna del progetto;
* *mercato* è un set di semafori dove ogni semaforo è inizializzato a 1 e riguarda il modo in cui è costruita la memoria corrispettiva (merce \*shm\_mercato[SO\_PORTI] --> shm\_mercato[i][j] farà riferimento alla merce di tipo j del porto i);
* *gestione* è un set che di norma prevede 1 solo semaforo, inizializzato a **SO\_NAVI+SO\_PORTI**, che ha il compito di gestire la sincronizzazione tra tutti i processi. Navi, porti e master eseguono una wait-for-zero sul semaforo, navi e porti lo fanno dopo averne decrementato di uno il valore;
* *dump* è un set contenente 1 semaforo per la sezione delle merci e 1 per la sezione delle navi, entrambi con valore 1, in quanto sono a rischio inconsistenza (potendo più processi scrivere nelle stesse aree).

CODA MESSAGGI:

È definita senza MSG\_COPY, ma le msgrcv vengono effettuate con **IPC\_NOWAIT**.

La coda contiene messaggi del tipo richiesta. Inoltre, per essa, abbiamo deciso di inserire come *mtype* l’indice del porto che effettua la richiesta. Ciò permetterebbe di avere *mtype* uguale a 0 e, non essendo ciò permesso, abbiamo implementato le funzioni invia\_richiesta e accetta\_richiesta in modo tale che effettuino esse stesse l’incremento e il decremento di 1 al tipo dei messaggi, senza che debbano preoccuparsene i processi chiamanti.

I SEGNALI:

SIGALRM viene lanciato dal master nel *main*, il segnale viene gestito dal signal\_handler, il quale lancia SIGUSR1 ai processi figli, nel caso in cui la simulazione non sia ancora da considerarsi terminata. Una volta finito il ciclo principale della simulazione, il processo master lancia i SIGUSR2 ai processi figli, generando il dump di terminazione del programma e la distruzione di tutte le risorse.

1. **Spiegazione del progetto**

La simulazione si svolge come segue: il processo master crea le risorse IPC, settando i dettagli (*ton/lotto* e giorno di *scadenza* del tipo di merce) di ogni lotto e la posizione di ogni porto nelle rispettive shared memory.

Dopodiché genera il vettore dei parametri da passare ai processi figli e inizia a eseguire le fork e le execve; fatto ciò, si ferma su un semaforo in **wait-for-zero**. Ogni processo decrementa di 1 il valore del semaforo di gestione, segnalando così al master di essere pronto. Quando tutti i processi sono pronti inizia il ciclo di dump in cui il master esegue la raise di **SIGALRM**, 1 secondo alla volta per eseguire il dump giorno per giorno.

Nel frattempo, i processi porto e nave partono settando il valore dei parametri in ingresso passati loro attraverso la execve e si agganciano alle risorse IPC, dopodiché:

* il *porto* inizializza casualmente la propria banchina e genera le proprie merci (assegnategli arbitrariamente in quantità **SO\_FILL/SO\_PORTI**), inserisce offerte e richieste sulla *shared memory* *mercato* e invia le richieste alla coda per poi decrementare di 1 il semaforo di gestione e mettersi anch’esso in wait-for-zero;
* la *nave* si aggancia alle risorse IPC già create e, dopo aver superato la richiesta di wait-for-zero sul semaforo di gestione, parte a tutti gli effetti chiamando la funzione codice\_simulazione.

La simulazione vera e propria parte a questo punto. La nave genera una posizione casuale sulla mappa (quella di inizio simulazione) e da lì calcola in che porto andare. Una volta nel porto, tenta di accedere al mercato (tramite il semaforo che ne determina la mutua esclusione sulle risorse che riguardano il porto di attracco) e, non appena riesce, inizia a cercare la prima richiesta soddisfabile nella coda messaggi.

Una volta accettata la prima richiesta, si è definito il porto di destinazione. Quindi cerca le altre richieste che lo riguardino (fino a un massimo di *reqlett* confronti) e fa le dovute verifiche sulle scadenze e il peso delle merci per stabilire quali merci caricare. A tal punto parte e, una volta attraccata nel porto di destinazione, scarica le merci (verificando che non siano scadute nel frattempo, in tal caso vengono contrassegnate come *scadute\_in\_mare*) e fa ripartire il ciclo di carico finora descritto.

**NOTA:** Le navi non ricevono il segnale di SIGUSR1 finche si trovano nelle fasi di carico/scarico e di viaggio (funzione attesa). Questa scelta è stata presa per limitare le attese di una nave carica e massimizzare il tasso di successo dei trasporti che effettua. La maschera del segnale di SIGUSR1 viene gestita e ripristinata alla fine della funzione sopra citata.

1. **Pro**

* La simulazione tenta di non far sprecare viaggi alle navi, questo significa che è stata limitata in maniera più specifica la scadenza delle merci in mare, grazie alle scelte implementative riguardanti il ciclo di carico delle merci.
* La memoria ***shm\_mercato***, che definisce quali merci sono presenti in un porto, è gestita come una matrice. Si ha così una migliore capacità di riferirsi a una determinata merce di un dato porto. Ne consegue una ricerca più veloce e semplice.
* Abbiamo deciso di inserire una coda di messaggi per le richieste dei porti in modo da facilitare il mantenimento della consistenza dei dati. Essendo gestito con un semaforo il mercato, le offerte verranno controllate, mentre le richieste non rischieranno problemi di concorrenza tra navi per la natura “atomica” dei messaggi.

1. **Contro**

* Di contro all’efficienza con cui le navi spostano le merci, c’è il fatto che molte rischiano di scadere in porto, soprattutto nel caso in cui le date di scadenza siano prossime all’inizio della simulazione.
* La memoria usata per gestire le risorse dei porti (***shm\_mercato***) è gestita come una matrice, ciò comporta che all’aumentare di ***SO\_MERCI*** e di ***SO\_PORTI*** e al diminuire di ***SO\_FILL*** lo spreco di memoria diventi notevole, in quanto si ha memoria inutilizzata a definire quali merci non sono attualmente presenti in un porto.