RELAZIONE PROGETTO DI SISTEMI OPERATIVI

Studenti Actis Nathan (mat. 985578), Gioana Renato (mat. 963518)

Introduzione:

Per semplificare la comprensione della seguente relazione verranno esaminate per prime le strutture condivise da ciascun processo per poi entrare nel dettaglio sul funzionamento dei singoli processi.

Shm.h:

L'header *"shm.h"* contiene alcune macro e tutte le struct necessarie al corretto funzionamento del progetto:

- TEST_ERROR: generico sistema di debugging (simile a quello visto negli esercizi) che scrive in output alcune informazioni relative a eventuali errori nel codice, con riferimento alla linea immediatamente precedente alla chiamata di *TEST_ERROR*.

- LOCK_BAN(ID)/UNLOCK_BAN(ID): regolano l'accesso alle banchine del porto con id "ID" utilizzando il semaforo appartenente al set *bancid* con *sem_num* == ID.

- DISTANCE(a, b): calcola la distanza tra due punti sul piano prendendo come parametri due set di *struct coordinates.*

- Struct coordinates: semplice struct composta di due campi di tipo double *x* e *y*, utilizzate per rappresentare la posizione di navi e porti sulla mappa.

- Struct my_msg_t: struct utilizzata per la costruzione di messaggi da inviare sulle message queues, composto dal *long int id*, rappresentante il type del messaggio e contenente l'id di una nave o porto a seconda della coda in cui viene spedito il messaggio, e da un *pid_t* pid contente il relativo pid.

- Struct merce: struct rappresentante le merci che vengono generate e su cui navi e porti operando durante la simulazione, si compone di *int status*, rappresentante lo stato della merce (0: al porto, 1: su nave, 2: consegnato, 3:

scaduta in porto, 4: scaduta in nave), *int id* che rappresenta il tipo di merce, *int size*, il peso in tonnellate di una singola quantità di merce, *int vita*, la sua durata in giorni prima della scadenza, *int num*, il numero di lotti di tale merce, *int pre*, quanti di tali lotti sono attualmente prenotati da una nave.

- Struct nave: struct rappresentante una nave, composta da *int idn,* ossia l'id della nave, *struct coordinates coord*, la sua posizione sulla mappa, *int idp_dest* e *idp_part,* rispettivamente l'id del porto di destinazione in cui scaricare le merci e del porto di partenza da cui caricarle, *double carico* rappresentante il peso complessivo delle merci a bordo, *double carico_pre* rappresentante il peso complessivo di tutte le merci che la nave ha prenotato.

- Struct porto: struct rappresentante un porto, composto da *int idp, ossia l'id del porto, struct coordinates coord,* la sua posizione sulla mappa, *struct merce *ric* e *struct merce *off,* rispettivamente il vettore di merci in richiesta e il vettore di merci in offerta.

- Struct shared_data: struct per la creazione di una memoria condivisa necessaria per la comunicazione tra navi e porti, in particolare per la gestione delle merci, composta da *struct merce *merci* rappresenta i campioni di ogni tipologia di merce, *struct porti *porti* rappresentante tutti i porti sulla mappa.

- Struct dump, dump_2, dump_3: memorie condivise utili al fine della realizzazione del dump giornaliero e finale.

Nel caso particolare di *shared_data* verrano create due memorie condivise accessibili e navigabili tramite i puntatori *sh_mem* e *sh_mem_2*.

Var.h

L'header *var.h* contiene tutte le variabili descritte nella consegna più alcune nuove necessarie per il corretto funzionamento della simulazione, sfrutta il metodo *setvar()* per riempire tali variabili con i numeri contenuti nel file *variabili.txt.*

Le nuove variabili dichiarate sono:

- MERCI_RIC_OFF: rappresenta il numero di merci generabili in richiesta/offerta di ciascun porto ogni giorno (lunghezza dei vettori *ric* e *off*).

- MERCI_RIC_OFF_TOT: rappresenta il numero complessivo di merci ottenuto moltiplicando MERCI_RIC_OFF per il numero di giorni.

Master.c

master.c è il processo principale del progetto, incaricato di creare navi, porti, meteo e tutti gli IPCs necessari al corretto funzionamento del progetto, ossia:

- Struct shared_data sh_mem: si occupa dei porti (ma non delle merci presenti in essi) e dei campioni di merce da usare come "stampo" per la generazione delle merci.
- Struct shared_data sh_mem_2: si occupa delle merci presenti nei porti e sulle navi.
- Struct my_msg_t msgN: message queue per i messaggi che riguardano le navi.
- Struct my_msg_t msgP: message queue per i messaggi che riguardano i porti.
- Struct dump* dmp, dump_2* portsh, dump_3* merck : memoria condivisa utilizzata per il dump giornaliero e finale.
- Struct sembuf sops: vettore di 4 semafori numerati da 0 a 3: (0: protezione sh_mem, sh_mem_2 e dump, 1: creazione dei porti in ordine, 2: controllare la terminazione di ogni processo figlio, 3: far partire navi e meteo contemporaneamente)

Il metodo main() di master.c si occupa, oltre alla generazione di porti, merci, meteo e IPCs, anche di tener conto del trascorrere del tempo nella simulazione richiamando ogni secondo una *nanosleep()* (della durata stessa di un secondo), al termine della quale lancia un SIGALRM che, gestito dall'handler *alarm_giorni*, invia a porti, navi e meteo un SIGUSR1 per avvisarli del trascorrere di un giorno, oltre a scrivere il dump giornaliero. Alla fine di SO_GIORNI lancia un SIGINT che attraverso l'handler *fine_sim* terminerà la simulazione e lancerà il dump finale.

Metodi ausiliari:

- void fine_sim(int signal): handler per il segnale SIGINT, termina la simulazione inviando SIGINT a porti, navi e meteo, quindi elimina tutti gli IPCs creati.

- void dump(): recupera tutte le informazioni relative al dump giornaliero e lo stampa.
- void resetSems(int sem_id): azzera tutti i semafori del vettore con id *sem_id*.
- void gennavi(): crea SO_NAVI processi nave.
- void genporti(): crea SO_PORTI processi porto.
- void genmeteo(): crea il processo meteo.

Porto.c

Il metodo *main()* del processo porto si occupa di creare il singolo porto posizionandolo sulla mappa, creando i vettori di merci relativi a richiesta e offerta e le banchine.

Metodi utilizzati:

- void handle_morte(int signal): handler per SIGINT, termina il processo allo scadere di SO_GIORNI giorni.
- void handle_time(int signal): handler per SIGUSR1, tiene conto del passare dei giorni facendo scadere le merci presenti al porto aggiornandone il tempo di vita rimanente. In più ogni giorno rigenera le merci in richiesta e offerta.
- void creaPorto(): posizionamento casuale del porto (dal quinto in poi) sulla mappa, impedisce la creazione di più porti con le stesse coordinate.
- void genric(): generazione delle merci in richiesta nel porto con tutti i relativi attributi.
- void genmerci(): generazione delle merci presenti al porto (in offerta) con tutti i relativi attributi.

Nave.c

La nave è il processo più complesso del progetto. Il suo metodo *main()* segue il seguente algoritmo di ricerca:

ricerca di un porto in cui consegnare (dest), se non trovo aspetto il prossimo giorno;

prenota tutte le merci richieste da quel porto fino a esaurimento capienza o richiesta;

ordina i porti in ordine di distanza dal porto a cui consegnare;

while(merci prenotate > 0){

    scegli il porto più vicino che offre almeno una merce richiesta (part);

    se nessun porto offre le merci richieste azzera le prenotazioni, altrimenti prenota merci da caricare;

    raggiungi porto di carico;

    carica merci;

    raggiungi porto di destinazione;

    scarica merci;

}

(Da ripetere fino al termine della simulazione).

Tale algoritmo è stato ritenuto una scelta migliore rispetto al cercare di recuperare tutte (o quante possibili) merci richieste da un porto per soddisfare al meglio la sue richiesta in quanto, data l'alta probabilità di tempeste, mareggiate e vortici, più tempo una nave passa in mare e più rischia di perdere il suo carico, al contrario l'algoritmo scelto cerca di garantire almeno un soddisfacimento parziale delle richieste al prezzo di aumentare il tempo totale di lavoro di ogni nave.

Il *main(),* inoltre, si occupa di comunicare al meteo quali navi possono essere vittime di tempesta o quali sono ferme in porto in caso di mareggiata.

Metodi ausiliari:

- void handle_morte(int signal): richiama il proprio id da eventuali code e termina il processo.

- int containsOff(int portoid, int merceid): ritorna l'indice del vettore off del porto con id portoid che contiene merceid, -1 se non presente.

- void handle_time(int signal): handler per SIGUSR1, tiene conto del passare dei giorni facendo scadere le merci presenti a bordo aggiornandone il

tempo di vita rimanente.

- void handle_storm(int signal): handler di SIGUSR2, se la nave subisce una tempesta (il segnale inviato dal meteo), aumenta il tempo di viaggio di SO_STORM_DURATION secondi.

- void handle_swell(int signal): handler di SIGWINCH, scelto come "SIGUSR3" in quanto è il segnale meno utilizzato, se il porto in cui la nave si trova subisce una mareggiata (il segnale inviato dal meteo), aumenta il tempo di lavoro di SO_SWELL_DURATION secondi.

- void swap(int *xp, int *yp): semplice swap di due int;

- void ordinaporti(struct coordinates coord): riordina i porti a distanza crescente rispetto alle coordinate coord.

- int containsRic(int portoid, int merceid): come containsOff ma sul vettore *ric*.

- int getpart(): ricerca del porto da cui caricare, ritorna l'id o -1 se non trovato.

- int getdest(): ricerca del porto a cui scaricare, ritorna l'id o -1 se non trovato.

- void gennave(): posizionamento della nave (vuota) su posizione casuale della mappa.

- carico(): carico di tutte le merci prenotate presso il porto attuale (part).

- scarico(): scarico di tutte le merci prenotate a bordo presso il porto attuale(dest).

Durante questo ciclo vengono comunicati al meteo gli id delle navi in viaggio e dei porti con navi attraccate, rispettivamente attraverso msgN e msgP.

Meteo.c

Il processo *meteo.c* si occupa di rendere più impegnativo il lavoro delle navi creando 3 tipologie di eventi casuali: *tempesta, mareggiata* e *maelstrom*.

Così come accade in *master.c* anche il *main()* di *meteo.c* predispone un timer indipendente per la gestione dei *maelstrom*, che vengono lanciati ogni SO_MAELSTROM ore tramite l'invio di un SIGINT.

Gli altri metodi utilizzati sono:

- void handle_morte(int signal): handler per SIGINT, permette la corretta terminazione del processo.

- void tempesta(): seleziona una nave casuale tra quelle in transito e aggiunge alla durata del tragitto il tempo di attesa SO_STORM_DURATION tramite l'invio di SIGUSR2.

- void mareggiata(): seleziona un porto casuale e aggiunge al tempo di lavoro di ogni nave presente in tale porto un tempo di attesa SO_SWELL_DURATION tramite l'invio di SIGWINCH, scelto in quanto segnale meno utilizzato dal sistema operativo.

- void handle_time(int signal): handler per SIGUSR1, tiene conto del trascorrere dei giorni e ogni giorno richiama *tempesta()* e *mareggiata()*.

Makefile

Il makefile permette la corretta compilazione di tutti gli elementi del progetto, digitare "make" per liberare il terminale e compilare, "make run" per eseguire.