Università degli Studi di Torino (2023), 1–2

ARTICLE TYPE

Relazione Progetto di Sistemi Operativi

Oskar Heise

Matricola: 976322, email: oskar.heise@edu.unito.it, Turno: T2

Consegna

Si intende simulare il traffico di navi cargo per il trasporto di merci di vario tipo, attraverso dei porti. Questo viene realizzato tramite i processi *master*, *mave* e *porto*.
Nella simulazione esistono diversi tipi di merce. Queste vengono generate presso i porti. Le navi nascono da posizioni casuali e senza merce e il loro spostamento viene realizzato tramite una *nanosleep*. I porti hanno una certa domanda e offerta e le proprie banchine vengono gestite tramite *semafori*.

master.c

Il processo Master è quello incaricato di creare gli altri processi e gestire la simulazione. Viene inoltre creato e inizializzato il semaforo nominato *semaforo_master*, indispensabile per la corretta sincronizzazione dei processi *nave* e *porto*. Prima della creazione di questi ultimi, vengono aperte delle risorse IPC, in particolare le memorie condivise e la coda di messaggi.
La creazione dei processi figli avviene tramite due cicli *for* distinti, ecco come funziona:

```
for(i = 0; i < so_porti; i += 256){  /*
    incremento di 256 ad ogni blocco*/
    for(j = i; j < i + 256 && j < so_porti;
         j++){
    sem_wait(semaforo_master);
        switch (pid_processi = fork()){
            case -1:
                fprintf(stderr, "Errore
                    nella fork() del Porto")
                    ;
                exit(EXIT_FAILURE);
                break;
            case 0:
                execvp("./porti", args);
                break;
            default:
                signal(SIGCHLD,
                    handle_child);
                pause();
                break;
        }
    }
    sleep(0); /*inseriamo uno sleep(0) tra
        un blocco e l'altro*/
    printf("Creazione processi porto: %d/%d
         \n", j, so_porti);
}
```

Subito dopo l'ingresso nel ciclo, il semaforo, inizializzato a 1, viene diminuito di valore, attraverso un'operazione di *sem_wait*. I processi vengono creati attraverso l'operazione di *fork()* e uno *switch* si occupa di compiere le varie operazioni in base al valore ritornato. I processi vengono creati un po' per volta, per evitare sovraccarichi della CPU.

Subito dopo la creazione dei processi Nave e Porto, mi occupo di creare i processi destinati alla gestione del meteo e del timer. Inoltre mi occupo di gestire i segnali necessari per la corretta creazione delle navi. Infine aspetto il messaggio che mi consentirà di deallocare tutte le risorse IPC e di terminare il processo.

porti.c

I porti vengono localizzati in una certa posizione nella mappa e gestiscono un numero casuale di banchine. I primi quattro porti vengono sempre generati agli angoli della mappa. Questo viene fatto attraverso *generatore_posizione_iniziale_porto()*, presente nella libreria personalizzata *header.h* di cui parlerò in seguito. All'inizio del codice dei porti mi occupo di generare tutte le informazioni necessarie per il loro funzionamento e di generare la loro merce, successivamente le inserisco nell'array di strutture presente nella memoria condivisa creata in precedenze. L'indice di ogni porto viene stabilito usando il PID in questa maniera:

```
shared_memory_porto[(getpid() - getppid())
    -1] = porto;
```

Nel processo viene infine gestito *semaforo_master* e incrementato di 1 attraverso l'operazione di *sem_post*.

navi.c

Le navi sono i processi più importanti dell'intero programma, ed è qui che gestisco la maggior parte delle operazioni del Progetto, ovvero:

- Creazione delle navi;
- Movimento delle navi;
- Scambio delle merci.

Una buona parte del codice è dedicata alla memorizzazione delle variabili utili per le statistiche intermedie e finali, e la prima cosa che faccio nel codice è proprio quella di resettarle. Successivamente, mi occupo di ricevere le diverse informazioni dalla memoria condivisa, in particolare le informazioni dei porti. Dopo, genero tutte le informazioni della nave, in particolare la posizione iniziale della nave, attraverso la funzione

generatore_posizione_iniziale_nave().
Una parte fondamentale del processo *nave* è quella della "partenza". Grazie all'utilizzo dei semafori, i processi vengono creati uno per volta e, una volta completata la loro "inizializzazione", questi attenderanno prima dell'inizio della simulazione vera e propria. Una volta creati inizializzate tutte le navi, il master manderà un messaggio alle navi che cominceranno a cercare il porto in cui sbarcare e a scambiare le merci con questi ultimi. Il tutto avviene all'interno di un ciclo for che terminerà una volta raggiunti i giorni previsti dalla simulazione.

meteo.c

Il processo meteo viene creato subito dopo la creazione delle navi e si occupa della generazione di eventi meteorologici che influiranno sui vari elementi della simulazione. Questi eventi vengono gestiti tramite dei *thread*, che vengono creati nel *main()* in questo modo:

```
0 /* creazione dei thread */
1 pthread_create(&threads[0], NULL,
      thread_function_tempesta, NULL);
2 pthread_create(&threads[1], NULL,
      thread_function_mareggiata, NULL);
3 pthread_create(&threads[2], NULL,
      thread_function_maelstrom, NULL);
```

Gli eventi sono tre:

- Tempesta: ogni giorno colpisce casualmente una nave in movimento, fermandola sul posto per tot ore;
- Mareggiata: ogni giorno colpisce casualmente un porto, fermando tutte le operazioni per tot ore;
- Maelstrom: ogni tot ore, una nave viene affondata e il suo carico viene disperso.

print.c

Il processo print viene creato successivamente a quello del meteo e si occupa della gestione dello scorrere del tempo e della visualizzazione dei report giornalieri e finale. I report non vengono mostrati sul prompt dei comandi, ma vengono scritti in un file di testo separato, chaimato *"report.txt"* di modo da poter essere consultati più tranquillamente. Il tutto viene gestito attraverso un ciclo for e una *sleep()*. Ogni secondo, viene infatti aggiornato il paramentro *shared_memory_giorni>giorni* presente in shared memory, viene stampato il report giornaliero e vengono aggiornate le scadenze delle merci presenti nelle navi e nei porti. Una volta terminata la simulazione, attraverso una coda di messaggi, il processo manda un messaggio al master, che verrà avvertito e procederà a deallocare tutte le risorse *IPC* prima di terminare anch'esso.

```
0 messaggio_id = coda_messaggi_get_id(MSG_KEY
      );
1 strcpy(messaggio.messaggio_testo, "FINE");
2 messaggio.messaggio_tipo = 1;
3 msgsnd(messaggio_id, &messaggio, sizeof(
      messaggio), 0);
```