Relazione sullo svolgimento dell'assignment 2, A.A. 2012/13

NOTE GENERALI e SCELTE DI PROGETTO

Per affrontare il problema ho scelto di intraprendere una strategia mista bottom-up/top-down, a partire dalla scrittura della pseudocodifica che dava una visione generale della soluzione del problema, la prima azione è stata di suddividere la struttura in moduli principali. In particolare il file 'benzinaio.c' con il codice eseguito dai benzinai, e il file 'macchina.c' con il codicce delle auto.
Pe la gestione della comunicazione e della sincronizzazione utilizzo i semafori, le code di messaggi e la memoria condivisa, attraverso il file 'sem_mem.c'. Questo gestisce la creazione e l'inizializzazione di queste strutture.

Una volta terminata la parte relativa alla gestione del comportamento di benzionai e auto, ho creato il file 'starter.c' che si occupa di eseguire gli altri file in modo sequenziale, per risolvere questo problema ho deciso di creare un unico eseguibile che genera tre figli e gli fa eseguire i vari codici, gestendo possibili errori.

Per migliorare la compattezza del codice ho deciso di implementare la libreria con il file 'tutto.h' e 'tutto.o'. Questo mi permette di definire una sola volta le funzioni che servono a tutti gli eseguibili, per definire tutte le costanti delle chiavi per le strutture dati, e per includere le librerie di sistema indispensabili per l'esecuzione del programma.

Infine ho creato il makefile per compilare tutti i file .h e .c nel modo corretto. In particolare questo prima crea la cartella bin, poi crea il file oggetto .o del file 'tutto.c' che deve essere linkato con tutti gli altri quattro eseguibili, evitando di ricompilarlo più volte. Dopo di che crea gli eseguibili 'sem_mem' 'benzinaio' 'macchina' e li mette nella cartella bin, mentre il file 'starter' viene messo nella stessa cartella del makefile, per facilitare l'avvio.

FORK E GESTIONE DEI PROCESSI

Per quanto riguarda la ramificazione totale nella creazioni dei processi possiamo dire che inizialmente il processo 'starter' crea un figlio che eseguire il codice 'sem_mem' per l'inizializzazione delle strutture. Se tutto va a buon fine viene creato un altro figlio che fa eseguire il codice 'benzinaio' e un terzo figlio per il codice 'auto'.
A sua volta il processo benzinaio crea un figli, ed entrambi andranno a gestire una pompa. Alla fine del loro codice il padre aspetta la terminazione del figlio gestendo i suoi errori.
Per quanto riguarda il processo auto questo crea n figli impostabili attraverso una costante nel file 'tutto.h', il padre di questi processi non diventa una macchina, ma aspetta la terminazione di tutti le auto figlie gestendo gli eventuali errori di ritorno di una di queste.
Il processo starter aspetta la terminazione del processo benzinaio e auto, gestisce i loro errori e poi rimuove tutte le strutture dati e termina.

PSEUDOCODICE

Macchine{

wait(idSemStazione)

wait(mutexPompaLibera)
if(pompa1 == TRUE){
	pompa1 = FALSE
	p=1}
signal(mutexPompaLibera)

if(p == 1){
	signal(idSemPompa1)
	---- ricevo benzina ----
	wait(idSemPienoFatto1)
	---- ricevo importo da pagare ----
	---- invio soldi da pagare ----
	signal(idSemSoldiInviati1)
	wait(idSemVattene1)
	wait(mutexPompaLibera)
	pompa1 = TRUE
	signal(mutexPompaLibera)
}else{
	signal(idSemPompa2)
	---- ricevo benzina ----
	wait(idSemPienoFatto2)
	---- ricevo importo da pagare ----
	---- invio soldi da pagare ----
	signal(idSemSoldiInviati2)
	wait(idSemVattene2)
}
}

Benzinaio{

```
wait(IdNumAuto)

while( numeroAuto > 0){
        signal(IdNumAuto)
        wait(idSemPompa_i)

        if(fineGiornata == TRUE){
                break
        }else{
                ---- metto benzina ----
                ---- invio importo da pagare ----
                signal(idSemPienoFatto_i)
                wait(idSemSoldiInviati_i)
                soldi = ---- ricevo soldi ----
                signal(idSemVattene_i)

                wait(idSemMutexCassa)
                cassa = cassa + soldi
                signal(idSemMutexCassa)

                wait(IdNumAuto)
                numeroAuto  =  numeroAuto -1
        }
}
signal(IdNumAuto)
signal(idSemPompa_j)
fineGiornata = TRUE

wait(idSemMutexCassa)
paga = cassa / 2
signal(idSemMutexCassa)
}
```

STRUTTURA SPECIFICA
La struttura dell'applicativo consta di alcuni file eseguibili che sono preposti a gestire alcuni aspetti specifici del problema.

MAKEFILE: contiene le specifiche per la compilazione dei sottostanti file.

STARTER.c: gestisce l'avvio e la terminazione di tutti i processi che compongono il programma.

TUTTO.c: contiene le funzioni utili a tutti i processi.

TUTTO.h: contiene i prototipi delle funzioni comuni, le costanti e le altre librerie.

SEM_MEM.c: crea e inizializza i semafori, la memoria condivisa e le code dei messaggi.

MACCHINA.c: crea i processi macchina e contiene il relativo codice dei processi.

BENZINAIO.c: crea i processi benzinaio e contiene relativo codice dei processi.