Alessandro Borsoi 20014853

Il programma si compone dei seguenti files:
selvaggi.c
upo_semaphore.h
Makefile

È possibile compilare il tutto col comando make clean selvaggi. La compilazione con gcc è fatta attivando i seguenti flag: -Wall -Wextra -std=c11 -pedantic -g e controllata con valgrind –tool=memcheck –leak-cheack=full.

Il file upo_semaphore.h non è altro che una libreria header-only che fa da wrapper alle funzioni sui semafori init, wait e post. La necessità di fare ciò deriva dal fatto che OS X (sistema operativo su cui ho fatto la maggior parte dello sviluppo) non supporta gli unnamed semaphore. La soluzione semanticamente più vicina per questo sistema operativo è risultata quella di usare la libreria Grand Central Dispatch (GCD) fornita da Apple. Le funzioni sono quindi state nominate upo_sem_init, upo_sem_wait e upo_sem_post ed il tipo del semaforo è stato nascosto in una struttura upo_sem_s. Il preprocessore fornirà quindi condizionalmente l'implementazione per il sistema operativo in uso (linux o osx).

In selvaggi.c si trovano il main, la funzione da passare ai pthread per gestire i selvaggi e quella per gestire il cuoco. In più è stata creata una struttura che contiene tutti i dati condivisi dai thread in esecuzione e la cui modifica è stata confinata all'interno di un upo_sem_wait(&mutex) e un upo_sem_post(&mutex).

La struttura dati condivisa shared_t contiene un semaforo mutex, per far eseguire la sezione critica ad un solo thread per volta, e un semaforo full, che ha lo scopo di bloccare i selvaggi dal mangiare se la pentola è vuota. Le variabili porzioni e pasti_da_effettuare contengono rispettivamente le informazioni passate come parametri M e NGIRI. La variabile N che serve per sapere quanti selvaggi creare non è necessario sia condivisa tra i vari thread. La variabile pasti_in_pentola è utilizzata per permettere al cuoco di controllare se la pentola è vuota. Si è preferito usare una variabile del genere per comodità al posto di controllare il valore del semaforo full. In generale si è scelto di non implementare una struttura di tipo FIFO per gestire i pasti (ma solo un intero) in quanto considerati equivalenti, ovvero ad un selvaggio basta sapere che ci sia almeno una risorsa disponibile per accederla. In più il cuoco riempie la pentola completamente quando è vuota in mutua esclusione dai selvaggi. Si suppone quindi che non sia importante l'ordine di accesso ai pasti. La variabile pasti_cucinati serve a tenere il conto di quante volte il cuoco ha cucinato, come da richiesta. La variabile cucina serve per mettere in loop il cuoco e interromperlo una volta che tutti i selvaggi hanno finito i pasti. id_selvaggio serve per dare un id progressivo ai selvaggi creati e poter stamparne il valore a run-time.

Il main inizia controllando che siano stati passati i parametri richiesti e quindi li assegna a tre variabili: selvaggi (numero di selvaggi N che dovranno essere attivati), porzioni (numero di porzioni M che possono stare in pentola) e pasti (numero NGIRI di pasti che ogni selvaggio deve fare prima di terminare).

A questo punto viene allocato spazio nell'heap per la struttura dati condivisa shared. Vengono inizializzati i valori ed in particolare viene chiamata upo_sem_init su mutex (a 1) e full (a 0).

Si procede quindi alla creazione del numero di selvaggi richiesti più un unico cuoco e si passa shared come argomento. Si richiede con una pthread_join che il main attenda che i selvaggi abbiano finito la loro esecuzione prima di continuare.

Le operazioni successive consistono nel dire al cuoco di smettere di cucinare impostando a 0 la variabile cucina; il main si assicura che il cuoco abbia cessato la propria esecuzione con una pthread_join; dopodichè stampa il numero di volte in

cui il cuoco ha cucinato e libera la memoria allocata prima di terminare.

Come detto, il cuoco è in loop continuo sulla variabile cucina. In questo modo verrà interrotto dal main quando tutti i selvaggi avranno finito di mangiare. L'esecuzione del codice del cuoco è sezione critica e deve escludere i selvaggi fino a che non ha finito di cucinare. Questo avviene solo nel caso in cui la pentola sia vuota; in caso affermativo viene incrementata la variabile pasti_in_pentola e il semaforo full fino al riempimento completo della pentola.

I selvaggi, una volta creati, si mettono in un loop finito dipendente da quanti pasti devono fare. Una volta effettuati, possono terminare. Se ci sono pasti disponibili possono decrementare il semaforo full. Se non ci sono altri thread in sezione critica possono accedere alla loro passando il wait sul mutex. Qui mangiano, decrementano i pasti nella pentola ed escono dal mutex. L'id del selvaggio, assegnato solo alla prima iterazione, è stato fatto all'interno della sezione critica in quanto agisce su una variabile condivisa.