Problema:

Una **biblioteca universitaria** dispone di N posti per la consultazione dei libri e M (M<N) postazioni multimediali per la ricerca dei volumi. Ogni studente che accede alla biblioteca effettua la ricerca di un volume, successivamente si reca dal bibliotecario per richiederne una copia ed infine occupa un posto per consultare il libro. Si supponga che di ogni libro esistano K copie. Nel caso in cui non ci siano posti per la consultazione disponibili o se le copie del libro scelte sono terminate, lo studente lascia la biblioteca.

**var:**

posti\_multimediali\_liberi: semaforo generico (:=M)

posti\_consultazione\_liberi: int (:=N) *//contatore inizializzato a N*

esito\_ricerca\_volume: boolean

esito\_disponibilità: boolean *//Var condivisa. Si intende sia la disponibilità di copie che la disponibilità di posti*

studente\_pronto: semaforo generico

bibliotecario\_libero: semaforo binario (:=1)

mutex, mutex2, mutex3: semafori binari (:=1)

array di struct { id\_libro: int

n\_copie: int (:=K) } libro

**Processi**: studente, bibliotecario

**studente ()**

begin

wait (posti\_multimediali\_liberi) *//attende una postazione multimediale libera per ricercare il libro*

wait (mutex)

esito\_ricerca\_volume = ricerca\_volume *//controlla se il volume è presente nella libreria*

if (esito\_ricerca\_volume == false) *//se non è presente il volume ricercato*

signal (mutex)

signal (posti\_multimediali\_liberi) *//libero il posto multimediale*

exit () *//lascio la libreria*

else

signal (mutex)

signal (posti\_multimediali\_liberi) *//libero il posto multimediale*

wait (bibliotecario\_libero) *//attende che il bibliotecario sia libero*

signal (studente\_pronto) *//va dal bibliotecario per verificare la disponibilità di posti e copie*

wait (risposta\_bibliotecario) *//aspetta la conferma del bibliotecario*

wait (mutex2)

if esito\_disponiblità == true { *//se ci sono posti e ci sono copie*

//occupa posto e consulta libro

wait (mutex1)

libro[id\_libro

].ncopie ++ *//quando va via incrementa la disponibilità di una copia del libro*

signal (mutex1)

wait (mutex3)

posti\_consultazione\_liberi ++ *//e incrementa il numero di posti liberi per consultare i libri*

signal (mutex3)

exit () *//va via*

}

else *//se non ci sono posti o non ci sono copie vai via*

exit ()

end

**bibliotecario (id\_libro)**

begin

repeat

wait (studente\_pronto) *//attende uno studente che ha già controllato se è presente il volume*

wait (mutex2)

if (libro[id\_libro].ncopie > 0 && posti\_consultazione\_liberi > 0) *//se la copia è disponibile e ci sono posti*

esito\_disponibilità = true; *//variabile condivisa con lo studente. Comunica esito positivo*

signal (mutex2)

wait (mutex3)

posti\_consultazione\_liberi -- *//assegna il posto allo studente e decrementa il numero di posti liberi*

signal (mutex3)

wait(mutex1)

libro[id\_libro

].ncopie -- *//da il libro allo studente e decrementa la disponibilità di copie*

signal(mutex1)

signal (risposta\_bibliotecario) *//sblocca lo studente*

else

esito\_disponibilità == false *//variabile condivisa con lo studente. Comunica esito negativo*

signal (mutex2)

signal (risposta\_bibliotecario) *//sblocca lo studente*

signal (bibliotecario\_libero) *//si mette a disposizione di un altro studente*

forever

end

**Tipo di concorrenza:**

Il tipo di concorrenza è sincronizzazione e comunicazione.

C’è comunicazione grazie all’utilizzo di variabili condivise

C’è sincronizzazione quando, grazie all’utilizzo di semafori di “blocco”, i processi si attendono a vicenda; sono state applicate delle barriere di sincronizzazione, cioè un processo non prosegue fintanto che non ha conferma da un altro processo. I processi, dunque, potrebbero essere schedulati indipendentemente l’uno dall’altro però vanno in un ordine che è indipendente dall’algoritmo di scheduling utilizzato.

**Processi o thread:**

Potrebbero essere sia dei thread che dei processi.

Nel primo caso, cioè nel caso in cui siano dei thread, occorre scrivere nel main del programma dei CREATE\_THREAD per creare i thread e dei JOIN\_THREAD per la gestione delle attese.

Nel secondo caso, occorre far uso della funzione fork che serve per generare questi processi come figli. Quindi il codice di studente e bibliotecario è il medesimo: in pratica questi 2 processi condividono lo stesso codice, ma ne eseguono solo una parte grazie a dei controlli effettuati sul pid dei processi figli. In alternativa, è possibile utilizzare la funzione exec.

**Starvation:**

In questa soluzione, non ci dovrebbero essere situazioni di starvation in quanto c’è una stretta comunicazione tra i vari processi per procedere nei propri compiti. Sono stati introdotti più controlli di quelli strettamente necessari proprio per garantirci che ci sia sempre sincronizzazione.