Problema:

N studenti devono sostenere un **esame universitario**. La commissione d’esame è composta da due docenti ed un assistente. Ogni studente si reca da uno dei due docenti per sostenere l’esame e, se lo supera, passa dall’assistente per la registrazione. Il numero di studenti in attesa di registrare l’esame non deve essere maggiore di K. Nel caso in cui ci siano già K studenti in attesa di registrazione, i docenti sospendono gli esami in attesa che il numero di studenti in attesa dia minore di K/2 + 1.

**var:**

docente\_libero[2]: array di semafori binari (:=1) //inizializzati a 1 affinchè i docenti, all’inizio, siano liberi

studente\_pronto: semaforo binario (:=0)

conclusione\_esame: semaforo binario (:=0)

risultato\_esame: boolean //Var condivisa. Contiene l’esito dell’esame, superato o non superato

studenti\_in\_attesa: int (:=0) //Var condivisa. Contatore per la coda degli studenti che devono registrare

K: int

studente\_voto: semaforo\_binario (:=0) //indica se uno studente è pronto per registrare l’esame dall’assistente

conferma: semaforo binario (:=0) //indica se l’assistente ha registrato l’esame allo studente

situazione\_K: boolean (:=false) //flag utilizzato per segnalare all’assistente che la coda è piena

permesso\_assistente: semaforo binario (:=0) //usato dall’assistente per comunicare che la coda è k/2+1

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

**Processi**: studente, docente, assistente

**studente ()**

begin

wait (docente\_libero[i]) //attende un docente libero

signal (studente\_pronto) //segnala la sua disponibilità al professore per sostenere l’esame

wait (conclusione\_esame) //attende che l’esame sia concluso

wait (mutex)

if risultato\_esame == false //esame non superato

signal (mutex)

exit () //va via

else

signal (mutex)

signal (studente\_voto) //segnala la sua disponibilità per la registrazione all’assistente

wait (conferma) //attende che la registrazione sia avvenuta

exit () //va via

end

**docente (i)**

begin

wait (studente\_pronto) //attende l’arrivo di uno studente pronto per sostenere l’esame

wait (mutex)

risultato\_esame = esegue\_esame //funzione che genera il risultato dell’esame

if risultato\_esame [i] == false //se lo studente non ha superato l’esame

signal (mutex)

signal (conclusione\_esame) //comunica risultato e risveglia lo studente

signal (docente\_libero[i]) //segnala la sua disponibilità per effettuare un nuovo esame

else //se lo studente ha superato l’esame

signal (mutex)

wait (mutex2)

if studenti\_in\_attesa < K //se la coda degli studenti non ha raggiunto K

studenti\_in\_attesa ++ //incrementa la coda

signal (mutex2)

wait (m3) situazione\_K=false signal (m3) //flag a falso

signal (conclusione\_esame) //comunica risultato e libera lo studente

signal (docente\_libero[i]) //segnala la sua disponibilità per effettuare un nuovo esame

else //se la coda degli studenti ha raggiunto K

signal (mutex2)

wait (mutex3)

situazione\_K = true //imposta il flag a vero

signal (mutex3)

wait (permesso\_assistente) //sospende gli esami, attende il permesso per riprenderli

wait (mutex2)

studenti\_in\_attesa ++ //una volta ricevuto il permesso, incrementa la coda

signal (mutex2)

//comunica\_voto

signal (conclusione\_esame) //comunica risultato e libera lo studente

signal (docente\_libero[i]) //segnala la sua disponibilità per effettuare un nuovo esame

end

**assistente ()**

begin

wait (studente\_voto) //attende uno studente pronto per registrare il voto

wait (mutex3)

if situazione\_K == true //se il professore ha bloccato gli esami perché si è riempita la coda

signal (mutex3)

repeat //registra esami fino a che la coda non raggiunge il valore K/2 + 1

wait (mutex)

studenti\_in\_attesa -- //decrementa la coda

signal (mutex)

//convalida\_esame

signal (conferma) //registra il voto allo studente e confermaglielo

until studenti\_in\_attesa < K/2 + 1

signal (permesso\_assistente) //raggiunti i K/2+1, permetti al prof di riprendere gli esami

else

signal (mutex3)

wait (mutex)

studenti\_in\_attesa -- //decrementa la coda

signal (mutex)

//convalida\_esame

signal (conferma)

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, docente ed assistente è il medesimo: in pratica questi 3 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.