Nome_____ Cognome _____N. di matricola (10 cifre) _____ Riga ____ Col ____

UNIVERSITA' DEGLI STUDI DI BOLOGNA - CORSO DI LAUREA IN INFORMATICA
PROVA SCRITTA DI SISTEMI OPERATIVI
ANNO ACCADEMICO 2020/2021
17 gennaio 2022

Esercizio -1: Essere iscritti su AlmaEsami per svolgere questa prova.

Esercizio 0: Scrivere correttamente nome, cognome, matricola e posizione in tutti i fogli prima di svolgere ogni altro esercizio. Scrivere esclusivamente a penna senza abrasioni. E' vietato l'uso delle penne cancellabili, della matita, dei coprenti bianchi per la correzione (bianchetto) e la scrittura in colore rosso (riservato alla correzione).
Il compito e' formato da tre fogli, sei facciate compresa questa. Le soluzioni che si vogliono sottoporre per la correzione devono essere scritte negli spazi bianchi di questi fogli. Non verranno corretti altri supporti.
E' obbligatorio consegnare il compito, e' possibile chiedere che esso non venga valutato scrivendo “NON VALUTARE” in modo ben visibile nella prima facciata.
Per svolgere questo compito occorre solo una penna e un documento di identità valido. La consultazione o anche solo la disponibilità di altro materiale comporterà l'annullamento del compito (verrà automaticamente valutato gravemente insufficiente).

**Esercizio c.1:** Scrivere il monitor multibuf che implementi un buffer limitato (MAX elementi) di oggetti di tipo T che implementi le seguenti procedure entry:

```
void add(int n, T objexts[]);
void get(int n, T objects[]);
```

La funzione `add` deve aggiungere al buffer gli `n` oggetti passati col parametro `objects`. La funzione `get` deve predere dal buffer in modalità FIFO i primi `n` elementi presenti nel buffer e copiarli negli elementi del vettore `objects`. Entrambe le funzioni devono attendere che vi siano le condizioni per poter essere completate: che ci siano n elementi liberi per la add, che ci siano n elementi nel buffer per la get. Non sono ammesse esecuzioni parziali: mentre attendono le rispettive condizioni nessun elemento può essere aggiunto o rimosso dal buffer.
La definizione del problema C.1 presenta casi di possibile deadlock? quali?

**Esercizio c.2:** Un servizio di message passing sincrono senza selezione del mittente prevede una API con due funzioni:

```
sasend(msg_t msg, pid dest);
msg_t sarecv(void);
```

La funzione `sarecv` restituisce il primo messaggio ricevuto da qualsiasi mittente ed è bloccante se non ci sono messaggi pendenti. la funzione `sasend` si blocca fino a quando il messaggio `msg` non viene ricevuto dal processo `dest`.
Dato quindi un servizio di message passing sincrono senza selezione del mittente implementare un servizio di message passing sincrono (standard, quello definito nel corso) senza fare uso di processi server.

**Esercizio g.1:** Considerare i seguenti processi gestiti mediante uno scheduler preemptive a priorità statica su una macchina biprocessore SMP:

```
P1: cpu 4 ms; I-O 4 ms; cpu 2 ms
P2: cpu 2 ms; I-O 4 ms; cpu 5 ms
P3: cpu 5 ms; I-O 3 ms; cpu 3 ms
P4: cpu 10 ms; I-O 1 ms
```

l'Input-Output avviene su un'unica unità. Per il processo P1 ha priorità massima seguito da P2, P3 e P4 in sequenza decrescente, le richieste di I/O sono gestite in ordine FIFO. Calcolare il tempo necessario a completare l'esecuzione dei 4 processi.

**Esercizio g.2:** rispondere alle seguenti domande (motivando opportunamente le risposte):
a) Perché viene usata la paginazione per implementare la memoria virtuale?
b) L'algoritmo del banchiere dato uno stato di allocazione delle risorse restituisce un valore binario: safe o non safe.
In quali casi il sistema operativo esegue l'algoritmo del banchiere? Cosa succede se il risultato è safe e cosa se il risultato è non-safe?
c) Fornire esempi di file system con allocazione contigua, e spiegare perché sarebbe inefficiente usare altri metodi di allocazione nei casi d'uso tipici di questi file system.
d) perché l'invenzione degli interrupt ha reso i sistemi operativi più efficienti?