**Illustrare i meccanismi visti nel corso con cui si passa ad eseguire codice del sistema operativo.**

Il passaggio da mod utente a modalità kernel si ha nelle seguenti situazioni:

interruzioni hardware: Quando, durante l’esecuzione di un processo viene “sentita” un’interruzione hardware, viene generata una “trap” che passa ad eseguire il codice associato all’interrupt ( interrupt handler). Viene quindi terminata l’istruzione macchina che era in corso, salvata in memoria lo stato della computazione interrotta ( registro PSW), si salta alla posizione in mem dove è contenuto l’handler e una volta gestito l’interrupt si torna ad eseguire il processo.  
trap e chiamate di sistema: In questo caso le “trap” possono essere viste come interrupt software e vengono trattate allo stesso modo rispetto agli interrupt. Ciascuna trap ha un ID che viene usato per cercare nell’interrupt vector l’indirizzo della prima istruzione del corrispondente INTERRUPT HANDLER. Le system call sono realizzate tramite le trap. Per effetto della trap si passa alla kernel mode e in questo modo una volta gestito l’handler all’istruzione di ritorno dalla trap viene ripristinata la PWD del prog interrotto e si torna alla mod user.

**B) Illustrare la relazione tra grado di multiprogrammazione ed utilizzazione della CPU.**

Il livello di utilizzazione della CPU e il grado di multiprogrammazione sono in stretta correlazione fra loro. Un elevato grado di multiprogrammazione può portare al fenomeno del trashing con un altissimo tasso di page fault. Questo fa degenerare l’utilizzazione della CPU perché passerà più tempo a utilizzare il disco per cui il sistema spende gran parte del tempo nella gestione dei page fault. Quindi non sempre un elevata multiprogrammazione fa utilizzare di più la CPU, anzi, con un grado di multiprogrammazione troppo elevato l’utilizzazione della CPU può solo peggiorare.

**C) Anche con riferimento all’apposita funzione Unix, spiegare in che cosa consiste la creazione di un nuovo thread.**

La creazione di un nuovo thread per quanto riguarda la funzione Unix, consiste nella chiamata di funzione pthread\_create della libreria POSIX thread che permette di creare, all’interno del processo corrente, un nuovo thread che esegue la funzione il cui puntatore viene passato come argomento alla create.

In questo modo avremmo un thread il cui stack comprende il record di attivazione della funzione main oltre a quello che viene passato dalla funzione pthread\_create. La memoria accessibile al programma è automaticamente condivisa fra i vari threads.

**D) Spiegare perché nello scheduling della CPU ha senso tener conto della durata dei CPU burst dei processi, e come lo si fa.**

tenere conto della distribuzione delle lunghezze dei CPU burst nella scelta del quanto di tempo permette già di favorire i processi con CPU burst brevi, senza che il S.O. debba identificare quali sono; anche senza conoscere tale distribuzione, li favorisce l’uso di code multiple con quanti di tempo diversi. Se il S.O. stima la durata del prossimo CPU burst facendo una media esponenziale dei CPU burst fatti nel passato sommatoria che va da i=0 a n di (alfa\*t(n-1)), ne può tenere conto nella scelta del processo da parte dello scheduler, favorendo la scelta di processi con valori bassi della stima. In tutti i casi (e maggiormente nell’ultimo) si anticipa in media il momento in cui terminano i CPU burst (come per la terminazione dei processi con SJF nei sistemi batch).

**E) Illustrare il significato delle operazioni down e up sui semafori. Spiegare che cosa si intende per “semaforo binario” e “semaforo contatore”. Spiegare inoltre che cosa si intende per “spinlock” e “futex”.**

Le operazioni di down e up sui semafori servono per aumentare e decrementare rispettivamente il valore dei semafori. In un semaforo binario avremmo solo la possibilità di incrementarlo da 0 a 1 o decrementarlo da 1 a 0. La differenza con il semaforo contatotore è che esso può avere valori maggiori di 1 e serve per gestire N risorse. Si potranno quindi applicare N down fino a far diventare il semaforo 0 e il processo che proverà ad accedere dopo questi rimarrà in attesa.

Lo spinlock è utilizzato principalmente per garantire la mutua esclusione e sono implementati tramite attesa attiva. Questa scelta è efficace solo se : i thread che li utilizzano sono in esecuzione su due diverse CPu. Mentre il thread in attesa attiva consuma cicli della cpu su un procio l’altro può continuare finchè non ha finito. La sezione critica è breve: sotto queste condizioni l’attesa attiva può essere vantaggiosa perché evita di effettuare un cambio di contesto sulla CPU su cui gira il thread che deve attendere.

I futex: con una singola istruzione tipo TSL si decrementa una variabile di lock leggendo il valore precedente: se era 1 libero, si va avanti senza effettuare chiamate di sistema che comportano il passaggio a stato kernel. Se no si fa una chiamata al kernel per sospendere il thread.

In uscita si fa una operazione atomica di incremento e test e solo se necessario (ci sono thread sospesi: la var era <0) si chiama il kernel per svegliarne uno. Quindi non si chiama il kernel se non c’è contesa per l’accesso alla sezione critica.

**F) Spiegare la differenza tra la sincronizzazione mediante operazioni su monitor e la sincronizzazione di POSIX threads con variabili condizione. Descrivere il problema del produttore consumatore e una sua soluzione per POSIX threads mediante le variabili condizione, specificando quali variabili si utilizzano per la sincronizzazione, e dettagliando le operazioni di sincronizzazione per “produrre” e per “consumare” un elemento (non è necessario descrivere in dettaglio quelle sulla struttura dati “buffer”).**

La differenza principale è che nei monitor la mutua esclusione delle operazioni è a cura del compilatore, mentre nella libreria dei pthreads va programmata con i mutex e le operazioni lock/unlock e passando il mutex alla pthread\_cond\_wait, oltre alla condizione. Inoltre nei pthreads c’è l’operazione broadcast sulle condizioni pthread\_cond\_broadcast(&cond,&mutex) e i pthreads risvegliati dall’attesa sulle condizioni competono fra di loro (in caso di broadcast) e con i pthreads in attesa in lock per avere la mutua esclusione, per cui è opportuno usare un while e non if per verificare la condizione booleana sotto la quale bisogna fare wait.

**H) Confrontare la gestione di memoria con allocazione contigua e la paginazione dal punto di vista del costo in spazio per effettuare la traduzione degli indirizzi.**

nel caso dell’allocazione contigua, due indirizzi per processo (i valori per i registri base e limite quando il processo è il processo in esecuzione). La tabella delle pagine nel caso della paginazione, quantificandone la dimensione in funzione della dimensione delle pagine e della dimensione del processo

**Illustrare perché nei sistemi timesharing è opportuno prevedere la transizione dallo stato running (in esecuzione) allo stato ready (pronto per l'esecuzione) e in conseguenza di quale evento e con quale meccanismo si verifica**

• L'evento è lo scadere del quanto di tempo

• Il meccanismo è l'interruzione da timer

• Il motivo per cui esiste la transizione è impedire che un processo CPU bound monopolizzi una CPU: se il processo non effettua operazioni sospensive e non avvengono interruzioni da altri dispositivi, l'interruzione da timer è l'unico modo per far intervenire il sistema operativo e dare la CPU ad un altro processo. In assenza di questo, nè l'utente proprietario del processo nè altri utenti (se presenti) avrebbero la possibilità di far girare altri processi (su quella CPU, o in assoluto in caso di una sola CPU)

**Illustrare che cosa si intende per multiprogrammazione, per che motivo la si è introdotta (in particolare, quale vantaggio offre) e quali altri servizi sono richiesti al sistema operativo a causa della sua introduzione.**

Per multiprogrammazione si intende la presenza di più programmi caricati in memoria, almeno parzialmente. In questo modo si può passare dall'esecuzione di un processo p1 a quella di un altro p2 in un tempo sufficientemente basso (nel caso in cui p1 debba attendere una operazione di I/O, in un tempo sensibilmente più basso di quello necessario per completare l'operazione); così si può sfruttare meglio la CPU facendo girare p2 nel frattempo. É stata introdotta proprio per migliorare l'utilizzazione della CPU. Richiede la gestione di questi passaggi (context switch, in teoria possibile anche caricando e scaricando l’immagine del processo da disco, ma molto meno conveniente), la traduzione degli indirizzi e la protezione di un processo da accessi accidentali o voluti da parte di un altro.

**Illustrare la politica di scheduling detta round robin, e indicare se è adatta per sistemi batch motivando la risposta.**

La politica di scheduling detta round robin realizza il timesharing mantenendo una coda dei processi pronti; il primo della coda è il processo in esecuzione, allo scadere del quanto di tempo viene inserito in fondo, la CPU assegnata al successivo. Nei sistemi batch non è necessario il timesharing e quindi i context switch in corrispondenza delle interruzioni da timer sono inutili.

**All’interno di una operazione down su un semaforo, un processo può venire sospeso. Allora in che senso si dice che l’operazione deve essere atomica (indivisibile)? Spiegare che cosa in effetti non deve essere diviso, con particolare riferimento al caso di due processi che chiamano (quasi) contemporaneamente down sullo stesso semaforo. Quali meccanismi si possono usare per ottenere l’atomicità per l’esecuzione di down e up e perché questi meccanismi non sono una buona soluzione generale al problema delle “corse critiche” ma sono accettabili per l’implementazione di down e up?**

Non deve essere possibile, nel caso in cui il semaforo vale 1, che due processi trovino entrambi il valore del semaforo >0 prima che entrambi lo decrementino (quindi, almeno per uno dei due, test e decremento sono «divisi» da operazioni dell’altro) procedendo quindi senza sospendersi. Le operazioni possono essere realizzate come chiamate di sistema; su sistemi uniprocessore l’atomicità può essere realizzata disabilitando le interruzioni (accettabile perché fatto all’interno di una chiamata di sistema e non lasciandole disabilitare al programmatore); su sistemi multiprocessore con istruzioni come TSL, con attesa attiva, accettabile perché si deve attendere l’esecuzione di poche istruzioni macchina

**Spiegare che cosa si intende per semafori privati: per che motivo li si utilizza, e come (chi fa up e down e quando). Illustrare una soluzione generale per l’allocazione di risorse mediante semafori privati**

Li si utilizza per poter programmare una politica per scegliere quale processo mandare avanti tra un insieme di processi sospesi, mentre se tutti fossero sospesi su uno stesso semaforo, la scelta sarebbe demandata all’implementazione di up.  
Si inizializzano a zero.

Su un semaforo «privato» di un processo P, o di un insieme di processi, solo quei processi fanno down, quando devono attendere che una condizione booleana diventi vera (es. disponibilità di una risorsa) Altri processi fanno up quando risulta vera la condizione che il processo o i processi stavano aspettando e si è deciso di scegliere quel processo (o uno dei processi dell’insieme): ad esempio, una risorsa è disponibile e si è scelto di allocarla a P Illustrare una delle soluzioni viste nel corso.

**Illustrare se e come con la paginazione si ottiene, o si può ottenere, uno dei possibili scopi del sistema di gestione della memoria: proteggere un processo dagli altri, per evitare in particolare che un processo possa modificare i dati di un altro processo (eccetto in caso di esplicita richiesta di condividere memoria).**

Con la paginazione i dati di un processo A sono protetti da accessi (per errore o per malizia) da parte di un processo B – tranne nel caso in cui il codice dei processi sia progettato per condividere parte dei dati, con esplicita richiesta al S.O. - perché la traduzione degli indirizzi fa sì che B possa accedere soltanto ai frame della RAM che sono stati allocati a B. Questo perché la tabella delle pagine è gestita dal sistema operativo; usando, a partire da un indirizzo di pagina fisica assegnata a B, un offset compreso fra 0 e 2^n -1 (dove n bit sono usati per l’offset), si arriva a un indirizzo fisico di una pagina di B. Se B tenta di accedere ad un proprio indirizzo virtuale che non corrisponde a spazio allocato al processo (una pagina per la quale, ad esempio con una tabella delle pagine a più livelli, non c’è un elemento della tabella), si ha un errore.