Consumer = legge nel buffer
Producer = scrive nel buffer
Buffer = struttura dati, un grande array con puntatore all'ultimo elemento e va sincronizzato l'accesso a questa zona

Sincronizzare = Dato da due elementi:

1. Sincronizzazione di basso livello per la zona di memoria —>Accesso in mutua esclusione alla risorsa (non possono esserci due processi che accedono alla stessa risorsa)
2. Sincronizzazione di più alto livello tra i processsi ovvero se il BUFFER è vuoto non devo abilitare il consumer, se il BUFFER è pieno non devo abilitare i producer.

Questi due elementi corrispondono ai **due semafori:**

1. Un semaforo s = che rappresenta la mutua esclusione x l'accesso di memoria dove risiede il buffer.
2. Un semaforo n = regola la semantica di alto livello, nel buffer non devono entrare consumer quando il buffer è vuoto, non devono entrare producer quando il buffer è pieno. I producer non appena finiscono di inserire il loro pezzo nel buffer, sbloccano i consumer bloccati dal buffer vuoto. Li sbloccano con il semSignal(n). A quel punto i consumer si competono l'accesso al buffer con il semaforo s.

*Per i producer se buffer infinito non c'e' bloccaggio. Ma se invece il buffer ha un limite devo aggiungere un semaforo che blocchi l'accesso dei producer quando il buffer è pieno. Esso viene sbloccato dai consumer quando viene liberata una zona.

Semaforo Binario:
Semaforo che può assumere il valore 0 o 1.
S = gestisce mutua esclusione
delay = il semaforo binario

Con il semaforo binario non si sa quanti elementi son nel buffer, serve quindi una variabile n che ne tenga il conto.
Il consumer prende il numero di elementi che son nel buffer n, lo decremento di uno perché ne ha consumato uno e inserisco il valore nuovo di elementi in nuova variabile d'appoggio m. Se m vale 0 e quindi non c'e' nulla nel buffer allora il consumer si mette di nuovo in attesa.

**Se invece di usare una variabile di appoggio m, faccio semplicemente il decremento con n- - —> SBAGLIATO perché il controllo if(n==0) è al di fuori della zona di controllo del semaforo s : è fuori dalla semWait(s) e semSignal(s). Ciò che è all'interno della sezione critica viene eseguito tutto di seguito senza nessun intervento di producer o altri. Ma terminata la sezione critica il producer potrebbe invece riattivarsi e incrementare n e quindi tempo di fare il controllo if, n potrebbe assumere già un altro valore rispetto a quello calcolato precedentemente e questo porterebbe ad errori.

Monitors:
Mentre il semaforo ha un'interazione con lo scheduler del sistema operativo, il costrutto di monitor ha invece un livello più alto e quindi non c'e' dialogo.
E' un costrutto, una procedura per cui io devo chiamare il monitor.
Il monitor ha diverse procedure.
I processi fanno una chiamata al monitor e si mettono in coda per entrare e poter usare una delle procedure.
I monitor hanno delle variabili **condizioni** su cui effetto le funzioni cWait(), cSignal() = queste due system call non hanno però interazione con lo scheduler del sistema operativo. Non influenzo lo scheduler direttamente, ma indirettamente.
Quando esistono le condizioni di blocco (es. il buffer è pieno) , il producer si blocca ma in coda.
Da più garanzia su possibili errori di sincronizzazione come quelli del semaforo, perché gestisce la sua coda
cWait = entro in coda
cSignal = sveglio il primo in coda

Message Passing:
Ulteriore modo per mutua esclusione + comunicazione tra processi.
Funziona con le due funzioni = send , receive

Esistono diversi modi di implementare le funzioni
Receive:

- Bloccante = se chiamo receive e buffer di ricezione è vuoto mi blocco in attesa di ricezione
- Non bloccante = "" è vuoto e continuo con la prossima istruzione

Send:

- Bloccante =
- Non Bloccante =

Indirizzamento:

- Diretto = mando e ricevo messaggio con PID tot. , se PID si guasta devo creare nuovo processo con PID tot affinché funzioni nuovamente. = Si manda ad un
- Indiretto = mando e ricevo messaggio a processi di un gruppo, non definisco la lista dei processi, ma a processi che hanno aderito ad un certo gruppo - non so a chi di preciso sto mandando (mando ad una porta, non so chi c'e' dietro quella porta). La porta o mailbox funziona da separatore tra scender e receiver , se uno dei due si guasta allora è possibile rimpiazzarli senza vincoli. = Facilita gestione di problema guasti.
    - Statico = il gruppo non cambia
    - Dinamico = il gruppo si modifica nel tempo

Formato:
Fisso / Variabile, per lunghezza, contenuto etc.

Disciplina di coda:
FIFO - ordine per orario di ricezione
Priority - ordine per priorità

Possibili modalità per scambio di informazioni:

- Send Bloccante - Receive Bloccante = Rischio deadlock, impongono sincronizzazione stretta tra scender e receiver. Entrambe attendono che scender mandi mess e che receiver consumi messaggio.
- Send nonblocking - Receive Bloccante = Send dopo aver inviato il messaggio continua l'esecuzione, il Receiver invece si blocca in attesa di un messaggio da ricevere = **COMBINAZIONE MIGLIORE** e più utilizzata = send definisce l'occupazione di banda in un canale , senza blocchi utilizzo la banda in maniera ottimizzata, mentre nel receive utilizzo una parte minima della banda.
- NonBlock send- Nonblock receive = non molto utilizzata

Esempio. 5.20
La receive deve essere bloccante: il primo che entra riceve il messaggio, son sbloccato, entro in sezione critica. Quando il primo esce dalla sezione critica trasmette messaggio in mailbox con la send. Nel frattempo gli altri sono in attesa sulla receive perché è bloccante, quindi non possono proseguire se non consumano messaggio. Il primo che riceve messaggio mandato dal primo entrerà in sezione critica.

La send deve essere non bloccante, perché altrimenti sarei bloccato nell’attesa che qualcuno entri in sezione critica prima di mandare il messaggio.

*C’e’ bisogno quindi obbligatoriamente di un primo messaggio già impostato nella mailbox.

Esempio. 5.21