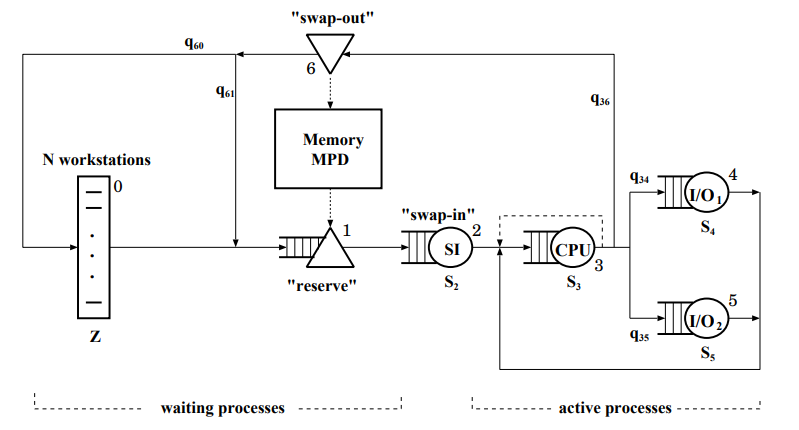
**Operating System Scheduling**



Il simulatore implementato riproduce una versione semplificata del reale protocollo di scheduling utilizzato nella maggior parte dei sistemi moderni. L'algoritmo simulato è Round-Robin con un relativo quanto di tempo.

***STAZIONI PRESENTI NEL MODELLO***

1. **Delay Station** (Infinite Server Station): questa stazione rappresenta appunto una stazione di ritardo per i vari jobs collegati al sistema. Possiede una distribuzione esponenziale negativa per quanto riguarda il tempo di servizio.
2. **Reserve Station**:questa stazione serve per limitare il numero di jobs “attivi simultaneamente” nel sistema (si tratta dunque di una stazione in cui tempo di servizio è distribuito secondo una distribuzione di probabilità sconosciuta).
3. **Swap-In Station**: una volta che il processo è ammesso nel sistema, questa stazione carica l’immagine del processo all’interno della memoria principale (il tempo di servizio è distribuito secondo la distribuzione esponenziale negativa).
4. **CPU**: stazione che rappresenta il core del sistema e che processa i vari jobs. In questo caso il tempo di servizio è distribuito secondo la distribuzione iper-esponenziale.
5. **IO1/IO2**: queste due stazioni rappresentano le stazioni di Input/Output il cui tempo di servizio è distribuito secondo la distribuzione esponenziale negativa.
6. **Swap-Out Station**: questa stazione serve per rilasciare la memoria adibita al processo.
7. In questo caso però, il tempo di servizio è trascurabile.

***EVENTI DEL SIMULATORE***

* ARRIVAL\_DELAY: arrivo di un processo alla Delay Station.
* ARRIVAL\_RESERVE: arrivo di un processo alla Reserve Station.
* ARRIVAL\_SWAP\_IN: arrivo di un processo alla Swap-in Station.
* ARRIVAL\_CPU: arrivo di un processo alla Cpu Station.
* ARRIVAL\_IO1: arrivo di un processo alla IO1 Station.
* ARRIVAL\_IO2: arrivo di un processo alla IO2 Station.
* ARRIVAL\_SWAP\_OUT: arrivo di un processo alla Swap-Out Station.
* DEPARTURE\_DELAY: partenza di un processo dalla Delay Station.
* DEPARTURE\_RESERVE: partenza di un processo dalla Reserve Station.
* DEPARTURE\_SWAP\_IN: partenza di un processo dalla Swap-In Station.
* DEPARTURE\_CPU: partenza di un processo dalla Cpu Station.
* DEPARTURE\_IO1: partenza di un processo dalla IO1 Station.
* DEPARTURE\_IO2: partenza di un processo dalla IO2 Station.
* DEPARTURE\_SWAP\_OUT: partenza di un processo dalla Swap-Out Station.

***INIZIALIZZAZIONE DEL SIMULATORE***

Prima di tutto vengono inizializzati tutti i vari contatori utilizzati dal simulatore.

Dopodichè, in base al parametro passato da riga di comando, viene lanciato il simulatore normale oppure una delle versioni semplificate dello stesso per scopi di validazione.

La Feature Event List (FEL) del simulatore è inizializzata inserendo i vari eventi (uno per ogni terminale connesso) di arrivo alla Delay Station (il cui il tempo di occorrenza corrisponde al tempo di Start del simulatore).

***GESTIONE DEI VARI EVENTI DEL SIMULATORE***

* **arrivalDelay**: gestisce semplicemente l’arrivo di un processo alla Delay Station, ritardando l’uscita dall’ultima del tempo di servizio attribuitogli.
* **departureDelay**: effettua l’inoltro del processo alla Reserve Station.
* **arrivalReserve:** gestisce l’ammissione del processo al sistema. Nel caso in cui il grado di multiprogrammazione risulti essere uguale al numero di processi nel sistema, il job corrente è inserito (enqueue) nella relativa coda secondo la politica FCFS (First Come First Served).
* **departureReserve:** effettua l’inoltro del processo alla Reserve Station.
* **arrivalSwapIn**: gestisce l’arrivo di un processo alla Swap-in Station, ritardando l’uscita dall’ultima del tempo di servizio attribuitogli. Nel caso in cui la stazione sia occupata, inserisce il processo nella relativa coda secondo la politica FCFS (First Come First Served).
* **departureSwapIn**: effettua l’inoltro del processo alla Cpu Station. Se presente, viene estratto (dequeue) un processo dalla relativa coda ritardando l’ultimo del tempo di servizio attribuitogli.
* **arrivalCpu**: gestisce l’arrivo di un processo alla Cpu Station. Vengono quindi aggiornati i tempi di servizio rimasto e il tempo di servizio corrente in base alla grandezza del quanto di tempo utilizzato dall’algoritmo di scheduling Round-Robin. Dopodichè, nel caso la stazione sia libera, viene ritardato il processo del tempo di servizio attribuitogli, viceversa viene inserito (enqueue) nella relativa coda secondo la politica FCFS (First Come First Served).
* **departureCpu**: gestisce la partenza dalla Cpu Station. Nel caso in cui il processo ha finito il suo tempo di computazione (CPU Burst), esso può essere inoltrato probabilisticamente ad una delle due stazioni di I/O oppure direttamente alla Swap-Out Station; viceversa, viene rimandato alla Cpu Station per terminare il suo tempo di servizio attribuitogli. Se presente, viene estratto (dequeue) un processo dalla relativa coda ritardando l’ultimo del tempo di servizio rimanente.
* **arrivalIO1**: gestisce l’arrivo di un processo alla IO1 Station, ritardando l’uscita dall’ultima del tempo di servizio attribuitogli. Nel caso in cui la stazione sia occupata, inserisce il processo nella relativa coda secondo la politica FCFS (First Come First Served).
* **departureIO1**: effettua l’inoltro del processo alla Cpu Station. Se presente, viene estratto (dequeue) un processo dalla relativa coda ritardando l’ultimo del tempo di servizio attribuitogli.
* **arrivalIO2**: gestisce l’arrivo di un processo alla IO2 Station, ritardando l’uscita dall’ultima del tempo di servizio attribuitogli. Nel caso in cui la stazione sia occupata, inserisce il processo nella relativa coda secondo la politica FCFS (First Come First Served).
* **departureIO2**: effettua l’inoltro del processo alla Cpu Station. Se presente, viene estratto (dequeue) un processo dalla relativa coda ritardando l’ultimo del tempo di servizio attribuitogli.
* **arrivalSwapOut**: gestisce l’arrivo di un processo alla Swap-Out Station.
* **departureSwapOut**: gestisce la partenza dalla Swap-Out Station, inoltrando probabilisticamente il processo alla Delay Station, se desidera ulteriore tempo di computazione oppure alla Reserve Station viceversa. Se presente, viene estratto (dequeue) un processo dalla Reserve Queue.

***PUNTO DI RIGENERAZIONE SCELTO***

Data la natura della quantità da stimare (la media teorica del tempo di permanenza nel “sotto-sistema attivo” da parte di un processo), l’identificazione del punto di rigenerazione del sistema è stata effettuata mediante l'approccio del metodo del “tagged customer for passage times”.

Il punto di rigenerazione da me scelto coincide quindi col verificarsi del seguente evento:

* **Arrivo del “tagged” job (n.0) job alla Cpu Station.**

Punto di entrata del sotto-sistema.

* **Il “tagged customer”DEVE aver già attraversato il sotto-sistema.**

Questo per evitare di aggregare inutili regeneration points (nel caso in cui il “tagged customer” non abbia attraversato almeno una volta il sotto-sistema, esso non accumulerebbe nessun valore).

* **Numero di jobs alla Reserve Station = 0.**

Questa scelta è motivata dal fatto che la distribuzione dei tempi di servizio è sconosciuta, fortemente influenzata dal grado di multiprogrammazione scelto e in parte anche dai tempi di attesa in ciascuna stazione dei terminali connessi al sistema. Quindi in generale essa non gode della proprietà di assenza di memoria.

* **Numero di jobs alla Cpu Station = 0.**

Essendo che la Cpu Station presenta una distribuzione dei tempi di servizio iper-esponenziale, essa non gode della proprietà di assenza di memoria.

Non avendo tempi di servizio distribuiti secondo l’esponenziale negativa (unica distribuzione di probabilità continua che gode della proprietà di assenza di memoria), una possibile situazione in cui, in almeno una delle due stazioni sia presente un job in esecuzione, non sarebbe un corretto punto di rigenerazione per il sistema poiché il tempo di servizio trascorso da questo ipotetico job influenzerebbe il tempo rimanente del medesimo nel successivo ciclo di rigenerazione. Si andrebbe quindi a violare la condizione di inter-indipendenza fra due cicli di rigenerazione adiacenti.

* **Numero di jobs alla Swap-Out Station = 0**
* **Numero di jobs alla Swap-In Station = CUSTOMER\_IN\_SWAP\_IN\_FOR\_REGENERATION\_COND**
* **Numero di jobs alla IO1 Station = CUSTOMER\_IN\_IO1\_FOR\_REGENERATION\_COND**
* **Numero di jobs alla IO2 Station = CUSTOMER\_IN\_IO2\_FOR\_REGENERATION\_COND**

Queste tre ultime stazioni hanno un variabile numero di clienti che definiscono il punto di rigenerazione per incrementare la frequenza di occorrenza dell’ultimo in base al modello del sistema scelto (modello originale o modello per validazione) e al numero di terminali connessi ad esso.

Nel caso in cui si abbia scelto il modello per la validazione, l’assegnazione dei valori a queste variabili è stata effettuata mediante l’utilizzo di MVA, per appunto estrarre il numero medio di clienti per stazione (approssimato all’intero più vicino) come euristica per velocizzare la ricerca di possibili punti di rigenerazione.

Viceversa nel caso in cui si abbia scelto il modello originale, si utilizza lo stesso approccio di assegnazione di valori alle variabili del caso precedente solo qualora il grado di multiprogrammazione sia maggiore o uguale al numero di terminali connessi al sistema (questo perché il modello sarebbe rilassato dal vincolo introdotto dalla Reserve Station proprio come nel caso precedente).

Nel caso invece si lanci il simulatore scegliendo il modello originale (in cui il grado di multiprogrammazione è in generale minore del numero di terminali connessi al sistema), il valore assegnato alle variabili sopracitate diventa:

* CUSTOMER\_IN\_SWAP\_IN\_FOR\_REGENERATION\_COND = 1
* CUSTOMER\_IN\_IO1\_FOR\_REGENERATION\_COND = 1
* CUSTOMER\_IN\_IO2\_FOR\_REGENERATION\_COND = 6

Questa scelta è motivata nuovamente dal fatto che, si è alla ricerca di quell’assegnazione di valori che massimizza la probabilità di occorrere in una condizione di rigenerazione per abbattere quindi i tempi dovuti all’attesa dell’evento (sempre in accordo con le proprietà che ogni punto di rigenerazione deve possedere per essere considerato tale).

Infine, è garantito che ciascun ciclo di rigenerazione possieda almeno un numero di osservazioni sufficienti (almeno 30) per essere sicuri di avere una somma delle ultime (variabile aleatoria) quasi-normale. Nel caso in cui ciò non avvenga, si provvede ad accorpare più cicli di rigenerazione in uno.

***FIRST STEP VALIDATION***

***BOTTLENECK ANALYSIS***

***SPECIAL FOUR CASES***

1. **time Slice δ = 3 msec; CPU burst with hyper-exponential distribution with parameters α = 0.8, β = 0.2, µ1 = 15 msec, and µ2 = 75 msec**

Sample Average Active Time = 4182.501631

Left Extreme = 4130.824114

Rigth Extreme = 4234.179149