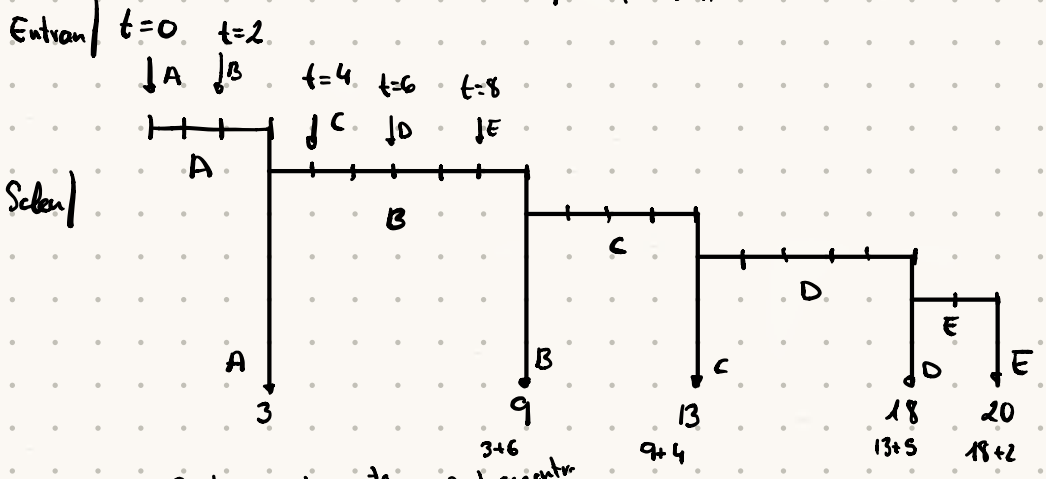
Proceso → nombre	Llegada → t. llegada	Servicio → tiempo servicio / tarea
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

FCFS (FIFO)

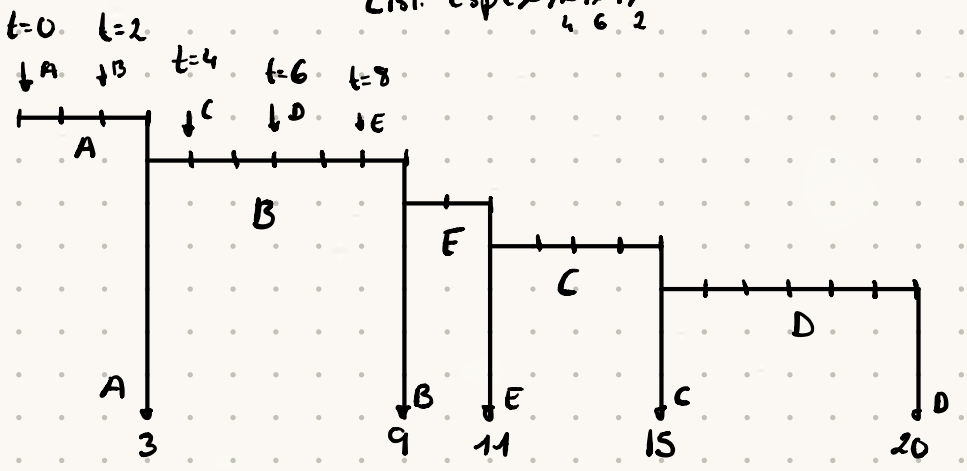
Lista Espera { B, C, D, E }



Proceso	$T_{Retorno}$ (Cuanto he pasado desde que entro hasta salir)	$T_{Servicio}$ (cuanto dura su ejecución)	T_{Espera} (Desde que entro hasta que empieza ejc)	$T_{Normalizado} = T_R / T_S$
A	3	3	0	1
	9-3		7-6	
B	7	6	1	7/6
			9-4	
C	9	4	5	9/4
D	12	5	7	12/5
E	12	2	10	12/2
			Medi: 4'6	2'5

SJF (El que menor tiempo ejecución)

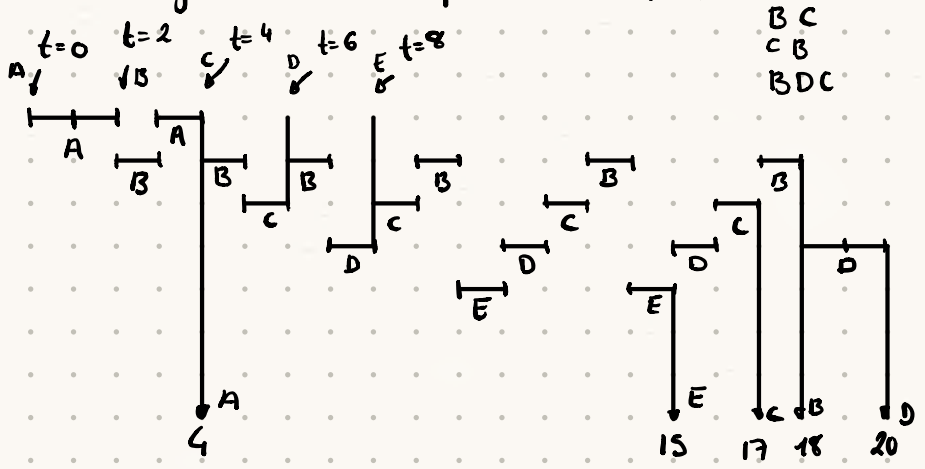
List. Esp (B, E, D, C)
4 6 2



Proceso	$T_{Retorno}$	$T_{Servicio}$	T_{Espera}	$T_{Normalizado} = T_R / T_S$
A	3	3	0	$3/3 = 1$
B	7	6	1	$7/6$
C	11	4	7	$11/4$
D	14	5	9	$14/5$
E	3	2	1	$3/2$
		Media:	3'6	1'84

Round-Robin (Cíclico, cada proceso un poco y pasa a otro)

Para rodaja de 1 u de tiempo : List. esp = { A, B, C, D, E }

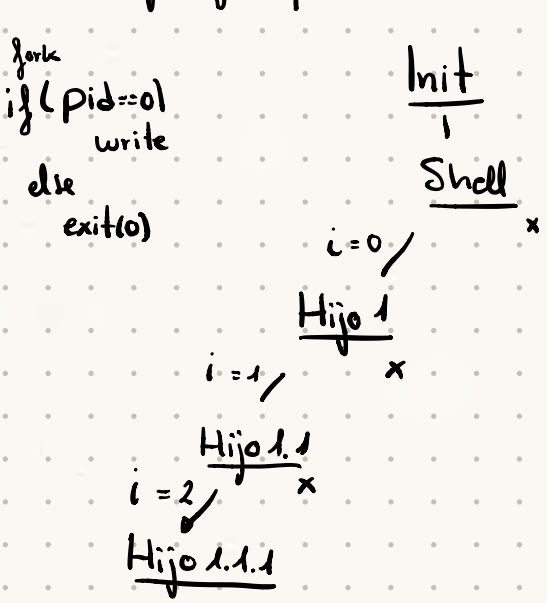


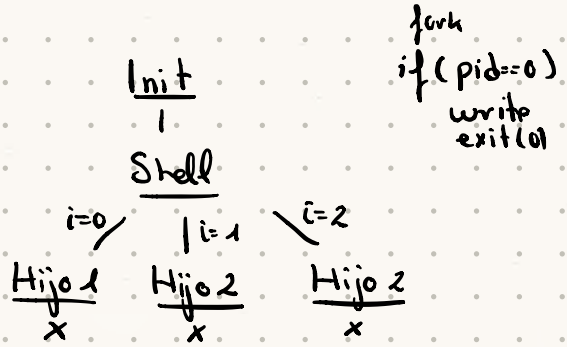
Cuando pasa al siguiente entra el anterior, pero si entra uno va antes.

¿Cuando entra en proceso en estado zombie?

Cuando el hijo muere y el padre no ha hecho wait, ya que si muere el padre lo hereda el init y este le da wait. La tabla se libera con un wait.

Dibujar jerarquía:





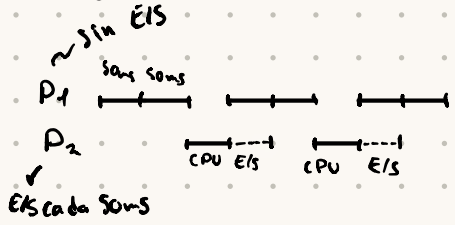
exit: Se muere el proceso.

kill: Envía una señal a un proceso para que muera.

¿Que transición no puede hacer un sist. con planif. no expulsivo?

Ejecutando a listo, no le pueden echar a lista de espera. Pero si puede pasar a bloqueado, ya que el mismo se suspende esperando a un evento.

Otro ejer:



... 100% de uso CPU

Cuando se bloquea para la E/S pasa a ejecutar otro

Mejor sist. planificación para multiproceso compartido:

Round - Robin, así todos tienen la oportunidad de ejecutar de forma equitativa.

Parcial 2014

```
int status, mod_pids;
for(int i=0, i<N, i++){
    pid = fork();
    if( pid!=0){ ~> Cuando es padre espera aquí mientras los hijos para empezar a devolver
        if( i==0){ ~> Si es el 1er padre cuando haga la operación termina el programa
            wait (&status);
            mod_pids = status + (getpid() % 10);
            printf("%d \n", mod_pids);
            exit (0);
        } else { ~> Si no es el padre original espera a los hijos y pasa a su padre la suma acumulada.
            wait( &status);
            mod_pids = status + (getpid() % 10);
        }exit (mod_pids);
    }else {
        printf("%d \n", getpid() ); ~ Los hijos imprimen su pid de verdad y no el 0.
    }
}
exit( getpid % 10); ~> El último hijo de todos, que origina la vuelta de todos.
```