Reporte

1. ¿Puede identificar una desventaja de la versión *preemptive* del algoritmo de *scheduling* propuesto? Si es así, ¿cómo la podría mejorar?

La primera desventaja que puedo identificar en el algoritmo propuesto, es el "starvation" que se puede generar, debido a que, ya en la cola ready, se elige a los procesos solo por prioridad ya sea por la preferencia dada ó por el PID, además luego de que un proceso es interrumpido vuelve a estar en ready, por lo que, los procesos con menos preferencia tienen menos opciones de ser seleccionados. Se puede mejorar, implementando una preferencia dinámica, donde a los procesos que llevan mayor tiempo de espera, se les sume preferencia para ser ejecutados. Una segunda desventaja que pude notar, es el tema de las condiciones de carrera que se pueden dar cuando dos procesos quieren acceder al mismo recurso, debido al mecanismo, los procesos muchas veces son interrumpidos antes de finalizar con el uso de un recurso, por lo que luego, si otro proceso quiere utilizar el recurso, y lo modifica, se podrían dar errores e inconsistencias, esto se puede mejorar con un aviso a los programas, cuando estén usando recursos compartidos y son modificados. La última desventaja que notó es lo que podría demorar todo el algoritmo al estar cambiando constantemente de proceso, esto creo que se podría mejorar ayudando con algún elemento la rapidez de los cambios.

2. ¿Cómo se podría extender su implementación para funcionar en un sistema *multicore* (> 1 núcleos)? ¿Podría disminuir alguno de los tiempos de los procesos (*turnaround*, *response*, *waiting*)? Describa qué decisiones de diseño debería tomar. No es necesario que lo implemente.

Se podría extender de la siguiente forma, al haber más de un núcleo se podrán ejecutar diferentes procesos al mismo tiempo. Entonces se tendrán los procesos con sus respectivos tiempos de espera y rafagas. Cuando un proceso pase a estado ready, ingresará a la cola de readys igual que en el algoritmo anterior, pero ahora se tendrá la posibilidad de tener en estado running a más de un proceso a la vez. Por lo que ante esta posibilidad, si es que un proceso(o más) ya está en running, eventualmente podría un proceso de la cola ready pasar a estado running en otro core, al mismo tiempo que un(os) proceso es(son) ejecutados en otro(s) core(s). Las decisiones de diseño que se tienen que tomar tienen relación con la organización espacial de los procesos, es decir, cómo asignar N procesos a M cores, existen diferentes posibilidades, tales como, usar un solo scheduling global, donde los procesos puedan ser asignados a cualquier core, otra forma es asignar una cantidad de procesos a cada core, y hacer scheduling en cada core con los procesos asignados ó podría ser un sistema mixto, bueno y cada scheduling tendrá su manera de asignar los procesos según su criterio. Todo esto podría disminuir el turnaround time, ya que la hora de término podría ser menos, el response time también podría disminuir ya que podría baja el tiempo en

ser atendido, y el waiting time, podría disminuir al poder ser más bajo el tiempo en ser atendido al estar en la cola ready.

3.¿Cuál versión del algoritmo se comporta mejor para un sistema interactivo? Proponga un caso de prueba con un conjunto de procesos *I/O bound* y compárelo en ambas versiones del algoritmo. Justifique el resultado.

Para mí se debería comportar mejor la versión non-preemptive, ya que, al ser un sistema interactivo, los tiempos de espera son largos, y las rafagas son cortas, por lo que, cuando entra un proceso al estado running, lo que tiene más sentido es que se ejecute completamente sin ser interrumpido y después pase al estado waiting para esperar la próxima rafaga. Como la rafaga es corta y el tiempo de espera es mayor, probablemente al ejecutarse cada rafaga sin interrupción no interferirá mayormente en los tiempos de los demás procesos que quieren ejecutarse y su waiting time probablemente disminuirá.

I/O bound process:Siguiendo el mismo formato que la tarea:

	P	I	N	A1	B1	A2	B2	A3	B3	A4
GERMY	3	6	4	1	4	3	6	2	5	2
RICHI	2	5	3	2	5	1	6	3		
CRISTIAN	1	4	3	1	6	3	5	2		

Resultados:

Preemptive, q = 2:	CPU	I	T	R	W
GERMY	5	,3,	25,	1,	17
RICHI	4	,2,	22,	0,	16
CRISTIAN	4	,2,	25,	0,	19

Non-preemptive:

GERMY	4	,0,	26,	1,	18
RICHI	3	,0,	23,	0,	17
CRISTIAN	3	,0,	17,	0,	11

En los resultados se puede apreciar, que la cantidad de entradas a CPU es menor en non-preemptive, el tiempo de turnaround time, es bastante parecido, sumando todos los tiempos, es menor el de non preemptive, el tiempo de response es el mismo, y en el waiting time, son bastante parecidos, aunque en la suma, el tiempo menor es de non-preemptive. Se

ve favorecido el proceso de menor preferencia en non-preemptive, mientras que los de mayor preferencia no tienen mayor cambio.