Examen 1º parcial de Sistemas Operativos Avanzados (11/6/2018)

No se permite el uso de documentación. Duración: 110 minutos.

Nombre y apellidos:

Tema 1 (2,5 puntos) Indique qué cuatro espacios de nombres crea el siguiente mandato (opciones *-u*, *-p*, *-m* y *-r,* respectivamente) y explique qué funcionalidad ofrece cada uno de ellos a los procesos descendientes del *bash* arrancado por dicho mandato: *unshare -u -p -m -r -f --mount-proc bash*

- opción -u: crea un nuevo espacio de nombres UTS que permite cambiar el nombre del host (mandato hostname) en el contexto del bash arrancado, de manera que todos los procesos descendientes del mismo van a ver ese nuevo nombre pero el resto de los procesos van a observar el nombre original de la máquina.

- opción -p: crea un nuevo espacio de nombres de PIDs que proporciona una numeración independiente para los identificadores de los procesos descendientes del bash. El propio bash tendrá el PID igual a 1 y a los procesos descendientes se les asignarán sucesivos PIDs.

- opción -m: crea un nuevo espacio de nombres de montaje que permite que un proceso descendiente del bash realice una operación de montaje (mount) que solo va a ser visible por todos los procesos descendientes del bash.

- opción -r: crea un nuevo espacio de nombres de usuario tal que el proceso bash creado pertenece al root (UID y GID igual a 0), incluso aunque el usuario que ejecutó el mandato sea un usuario normal.

Tema 4 (2,5 puntos) Identifique cuáles de las siguientes afirmaciones sobre el algoritmo de planificación CFS de Linux son ciertas, corrigiendo de forma razonada aquellas que sean falsas y explicando la motivación de las que sean verdaderas: **(a)** a los nuevos procesos se les asigna un *vruntime* de 0; **(b)** cuando se desbloquea un proceso, el planificador siempre considera como su *vruntime* el que tenía cuando se bloqueó; **(c)** el tamaño de la rodaja de ejecución de un proceso es el correspondiente a su peso, ya que cada peso tiene asignado un tamaño de rodaja determinado; **(d)** si hay varios procesos listos de diversos pesos y ejecutan todos 1 ms. y se bloquean, a todos ellos se les añadirá a su *vruntime* la misma cantidad; **(e)** si hay varios procesos listos de diversos pesos y ejecutan todos agotando su rodaja, a todos ellos se les añadirá a su *vruntime* la misma cantidad.

(a) Falsa. *A un proceso nuevo se le asigna como vruntime el mínimo vruntime entre los de los procesos listos para ejecutar. Si se le asignará un 0, el proceso acapararía el procesador hasta que su vruntime se fuera igualando con los de los otros procesos listos.*

(b) Falsa. *Si se usara siempre como vruntime de un proceso que se desbloquea el que tenía cuando se bloqueó, en caso de que haya estado mucho tiempo bloqueado, este proceso acapararía el procesador. Por ello, se usa el máximo entre el vruntime que tenía cuando se bloqueó y el mínimo vruntime entre los de los procesos listos para ejecutar menos una cierta cantidad de tiempo denominada sched_latency.*

(c) Falsa. *La rodaja de un proceso no viene determinada directamente por el peso del proceso sino que se tiene en cuenta también el peso de todos los procesos listos para ejecutar que existen en ese instante, repartiéndolo proporcionalmente. Solo por motivos didácticos, se incluye, a continuación, la fórmula correspondiente:*

*rodaja Pi = periodo * (Peso Pi / suma de los pesos de todos los procesos listos)*

(d) Falsa. *El valor que se añade al vruntime de un proceso cuando deja el procesador corresponde al tiempo que ha ejecutado, pero normalizado con respecto al peso 0 (para un proceso con peso 0, su vruntime corresponde al tiempo real usado por el proceso). Como todos los procesos han ejecutado 1 ms., la cantidad que se suma es distinta. Solo por motivos didácticos, se incluye, a continuación, la fórmula correspondiente:*

*vruntime Pi += 1 ms * (Peso 0 / Peso Pi)*

(e) Verdadera. *El valor que se añade a vruntime es el mismo para todos los procesos. Para entenderlo, hay que tener en cuenta que si un proceso tiene mayor peso que otro, conseguirá un tiempo de rodaja proporcionalmente mayor; pero para ese proceso, el tiempo "corre más lento" justo en esa misma proporción. De esta forma, el algoritmo es "justo" con todos los procesos porque asegura que todos ellos ejecuten el mismo tiempo "virtual". Solo por motivos didácticos, se incluye, a continuación, la fórmula correspondiente que muestra que el valor de ajuste es el mismo para todos los procesos listos, puesto que el peso específico de cada proceso desaparece.*

*vruntime Pi += rodaja Pi * (Peso 0 / Peso Pi) ; sustituyendo la rodaja por su valor, queda:*

*vruntime Pi += periodo * (Peso Pi / suma pesos listos) * (Peso 0 / Peso Pi) = periodo * (Peso 0 / suma pesos listos)*

Tema 2 y 3 (5 puntos) Dado un sistema multiprocesador con 2 procesadores, dibuje la traza de ejecución de ambos procesadores para los procesos A, B, C y D y para los siguientes escenarios, indicando si los procesos están en modo usuario o modo sistema, si hay cambio de contexto voluntario o involuntario y las diferentes rutinas de eventos que tienen lugar. Tenga en cuenta que:

- La rutina de interrupción de disco (*rutDisc*) tiene asociada una rutina de interrupción software de sistema denominada *rutSSistDisc.*
- La rutina de interrupción software de proceso de planificación se denomina *rutSProcPlan.*
- La rutina de interrupción software de proceso de terminación involuntaria se denomina *rutSProcTerm.*
- La rutina de tratamiento de la llamada a sistema *XXX* se denomina *rutXXX.*
- La rutina de tratamiento del fallo de página se denomina *rutFPag.*
- La rutina de excepción por división por cero se denomina *rutDivCero.*
- La rutina de interrupción entre procesadores (IPI) se denomina *rutIPI.*
- Sólo existen los procesos de usuario A, B, C y D. La planificación de los procesos se hace por prioridad, siendo el proceso A el más prioritario, a continuación el proceso B, después el C y finalmente el D.

1. Se trata de núcleos no expulsivos. Los procesos C y D son nuevos. El proceso A ejecuta una llamada *read()* sobre una tubería que está vacía, quedándose bloqueado. El proceso B ejecuta una llamada *wait()* sobre un semáforo que está cerrado, quedándose bloqueado. El proceso C trabaja en modo usuario durante un tiempo y después realiza una llamada al sistema *post()*, abriendo el semáforo por el que estaba bloqueado B. El proceso D ejecuta sólo código en modo usuario. Cuando el proceso B se desbloquea y es planificado, ejecuta una llamada *write()* sobre la tubería por la que espera el proceso A. A continuación, el proceso B ejecuta una llamada *read()* sobre un fichero. Dentro de la ejecución de la rutina de dicha llamada se produce un fallo de página. El proceso B se quedará bloqueado. FInalmente vendrá una interrupción de disco que despertará al proceso B, poniéndolo en ejecución de nuevo. Después de ser desbloqueado, el proceso A realiza una llamada *exit()*. El proceso C continúa en modo usuario. El proceso D, estando en modo usuario, realiza una división por cero, lo que provoca que el proceso finalice. El proceso B continúa en modo usuario.
2. El mismo caso anterior, pero para núcleos expulsivos.
3. Resuma los problemas de sincronización que nos encontramos en un sistema multiprocesador, indicando cómo se resuelven.