

PLANIFICACIÓN JUSTA: LOTERÍA Y LINUX COMPLETELY FAIR SCHEDULER

POR:

- ARMANDO UGALDE VELASCO
- DIEGO SANTIAGO GUTIÉRREZ

¿QUÉ ES LA PLANIFICACIÓN JUSTA?

- ALGUNOS PLANIFICADORES BUSCAN OPTIMIZAR MÉTRICAS COMO EL TIEMPO DE RESPUESTA O DE RECONVERSIÓN.
- SIN EMBARGO, SE PUEDE BUSCAR GARANTIZAR UNA REPARTICIÓN DEL CPU CON CIERTOS PORCENTAJES.
- SE LE CONOCE COMO PLANIFICACIÓN DE PARTES JUSTAS O PROPORCIONALES.

PLANIFICACIÓN DE LOTERÍA

- CADA CIERTO TIEMPO SE DEBE ESCOGER UN BOLETO PARA DETERMINAR QUÉ PROCESO VA A EJECUTARSE DESPUÉS
- LOS PROCESOS QUE DEBEN EJECUTARSE MÁS SEGUIDO DEBEN DE TENER MÁS OPORTUNIDADES DE GANAR LA LOTERÍA
- LOS BOLETOS SE UTILIZAN PARA REPRESENTAR LA PARTE DE CIERTO RECURSO QUE UNA ENTIDAD DEBE RECIBIR.



EJEMPLO

- SE TIENEN DOS PROCESOS: A Y B. A CUENTA CON 75 BOLETOS, MIENTRAS QUE B SOLAMENTE TIENE 25.
- EL PLANIFICADOR DEBE SABER CUÁNTOS BOLETOS EXISTEN EN TOTAL (100)
- SE DEBE ESCOGER UN BOLETO GANADOR (UN NÚMERO DE 0 A 99).
- A TIENE LOS BOLETOS 0 A 74, Y B LOS BOLETOS 75 A 99.
- EL BOLETO GANADOR DETERMINA SI A O B SE EJECUTA DESPUÉS.

63	85	70	39	76	17	29	41	36	39	10	99	68	83	63	62	43	0	49
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	---	----

- SE LOGRA UNA DISTRIBUCIÓN DE FORMA PROBABILÍSTICA (PERO NO DETERMINÍSTICA).
- B SOLAMENTE SE EJECUTA 4 DE 20 PARTES DEL TIEMPO (20%), EN LUGAR DEL 25% DESEADO.
- MIENTRAS MAYOR TIEMPO SE EJECUTEN ESTOS PROCESOS, AUMENTARÁ LA POSIBILIDAD DE QUE AMBOS LOGREN TENER LOS PORCENTAJES DESEADOS.



MECANISMOS DE BOLETO

MONEDA

- UN USUARIO REPARTE SUS BOLETOS ENTRE SUS PROPIOS PROCESOS.
- UTILIZA CUALQUIER MONEDA "INTERNA" QUE DECIDA.
- EL SISTEMA DESPUÉS CONVIERTE AUTOMÁTICAMENTE Dicha MONEDA AL VALOR CORRECTO GLOBAL.



EJEMPLO

- LOS USUARIOS A Y B TIENEN 100 BOLETOS CADA UNO.
- EL USUARIO A SE ENCUENTRA EJECUTANDO DOS PROCESOS: A1 Y A2, Y LE DA 500 BOLETOS A CADA UNO (DE 1000 EN TOTAL) EN SU MONEDA.
- B SE ENCUENTRA EJECUTANDO UN PROCESO, Y LE DA 10 BOLETOS (DE 10 TOTALES) EN SU MONEDA

- EL SISTEMA CONVIERTERÁ 500 BOLETOS DE A1 Y A2 A 50 BOLETOS EN LA MONEDA GLOBAL, PARA CADA UNO.
- EL SISTEMA LE ASIGNARÁ 100 BOLETOS CORRESPONDIENTES AL ÚNICO PROCESO DE B, EN LA MONEDA GLOBAL.
- LA LOTERÍA SE JUEGA EN LA MONEDA GLOBAL (200 BOLETOS EN TOTAL).

```
User A -> 500 (A's currency) to A1 -> 50 (global currency)
```

```
-> 500 (A's currency) to A2 -> 50 (global currency)
```

```
User B -> 10 (B's currency) to B1 -> 100 (global currency)
```

TRANSFERENCIA

- UN PROCESO LE “PRESTA” BOLETOS A OTRO.
- ÚTIL EN UN AMBIENTE CLIENTE-SERVIDOR: UN PROCESO CLIENTE LE PIDE A UN SERVIDOR QUE REALICE UN TRABAJO.
 - PARA ACCELERARLO, EL CLIENTE LE PRESTA SUS BOLETOS MIENTRAS ESTÁ PROCESANDO SU SOLICITUD.
 - CUANDO TERMINA, EL SERVIDOR TRANSFIERE DE NUEVO LOS BOLETOS AL CLIENTE.



INFLACIÓN

- UN PROCESO AUMENTA A SÍ MISMO EL NÚMERO DE BOLETOS QUE TIENE.
- NO ES ÚTIL EN UN AMBIENTE COMPETITIVO: UN PROCESO VORAZ PUEDE MONOPOLIZAR EL CPU.
- ÚTIL CUANDO LOS PROCESOS CONFÍAN ENTRE SÍ.



IMPLEMENTACIÓN DE LOTERÍA

- MUY SIMPLE
- SE NECESITA:
 - UN GENERADOR DE NÚMEROS ALEATORIOS.
 - UNA ESTRUCTURA DE DATOS PARA ALMACENAR LOS PROCESOS EN EL SISTEMA (LISTA).

- PARA TOMAR UNA DECISIÓN DE PLANIFICACIÓN, PRIMERO DEBE ELEGIR UN NÚMERO ALEATORIO (EL DEL BOLETO GANADOR) DEL TOTAL DE BOLETOS.
- ENTONCES, SIMPLEMENTE RECORREMOS LA LISTA UTILIZANDO UN CONTADOR PARA AUXILIARNOS EN ENCONTRAR EL BOLETO GANADOR.
- AL RECORRERLA, SUMAMOS LOS BOLETOS DE CADA PROCESO.



```
1 // counter: used to track if we've found the winner yet
2 int counter = 0;
3
4 // winner: use some call to a random number generator to
5 //           get a value, between 0 and the total # of tickets
6 int winner = getrandom(0, totaltickets);
7
8 // current: use this to walk through the list of jobs
9 node_t *current = head;
10 while (current) {
11     counter = counter + current->tickets;
12     if (counter > winner)
13         break; // found the winner
14     current = current->next;
15 }
16 // 'current' is the winner: schedule it...
```

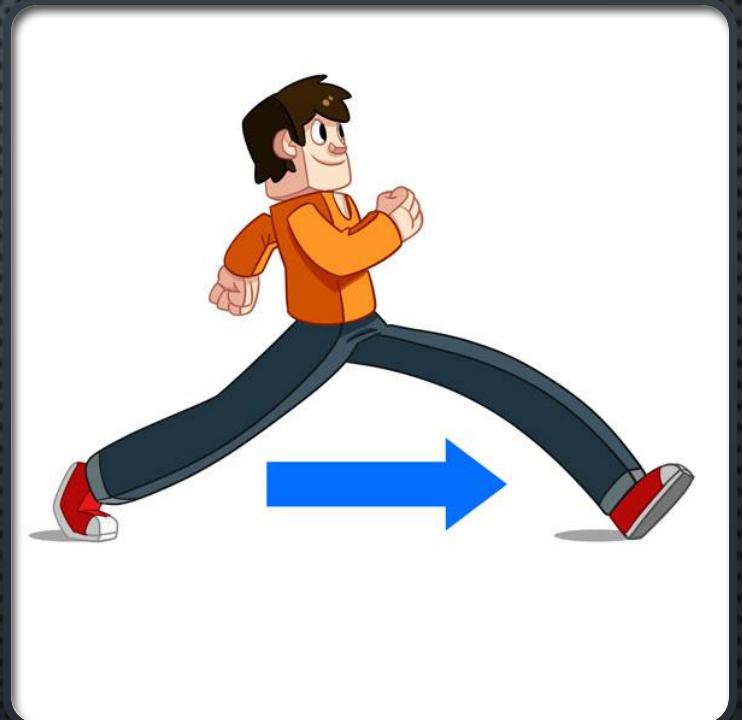
¿CÓMO ASIGNAR BOLETOS?

- ES UN PROBLEMA DIFÍCIL: EL COMPORTAMIENTO DEL SISTEMA DEPENDE DE LA FORMA EN LA QUE LOS BOLETOS SON DISTRIBUIDOS.
- UN ENFOQUE ES ASUMIR QUE LOS USUARIOS SABEN QUÉ HACER.
- EL PROBLEMA SIGUE ABIERTO.

¿POR QUÉ NO USAR UN ENFOQUE DETERMINÍSTICO?

- LA ALEATORIZACIÓN NOS PERMITE IMPLEMENTAR DE FORMA SENCILLA UN PLANIFICADOR APROXIMADAMENTE CORRECTO.
- OCASIONALMENTE NO PRODUCIRÁ PLANIFICACIONES EN LAS PROPORCIONES EXACTAS DESEADAS, ESPECIALMENTE EN PERIODOS DE TIEMPO CORTOS.
- **PLANIFICACIÓN DE ZANCADAS**, UN ALGORITMO DE PLANIFICACIÓN JUSTO Y DETERMINÍSTICO.

- CADA PROCESO TIENE UN **TAMAÑO DE ZANCADA** INVERSAMENTE PROPORCIONAL AL NÚMERO DE BOLETOS QUE TIENE.
- CADA QUE UN PROCESO SE EJECUTA, SE INCREMENTARÁ UN CONTADOR ASIGNADO A ÉSTE (CUYO NOMBRE ES **PASOS**) POR EL TAMAÑO DE SU ZANCADA PARA LLEVAR LA CUENTA DE SU PROGRESO GLOBAL.
- AL EJECUTAR UN PROCESO, SE DEBE ESCOGER EL QUE TENGA EL MENOR VALOR DE PASOS.



EJEMPLO

- A, B Y C, CON VALORES DE ZANCADA DE 100, 200 Y 40, RESPECTIVAMENTE, Y CON VALORES DE PASOS INICIALES DE 0.
- CUALQUIERA PUEDE CORRER, SUS VALORES DE PASOS SON IGUALES.
- A SE EJECUTA, SE ACTUALIZA SU VALOR DE PASOS A 100.
- B SE EJECUTA, SU VALOR DE PASOS SE ACTUALIZA A 200.
- C SE EJECUTA , SU VALOR DE PASOS SE INCREMENTA A 40.

- SE ESCOGE EL PROCESO CON EL VALOR DE PASOS MÍNIMO (C), Y SE EJECUTA
- SU VALOR DE PASOS AHORA ES 80.
- C SE EJECUTA DE NUEVO, AUMENTANDO SU VALOR A 120.
- A SE EJECUTA AHORA, AUMENTANDO SU VALOR A 200 (IGUAL AL DE B).
- C SE EJECUTA DOS VECES MÁS, AUMENTANDO SU VALOR DE PASOS A 160 Y LUEGO A 200.
- TODOS LOS VALORES DE PASOS SON IGUALES DE NUEVO, Y EL PROCESO SE REPITE DE FORMA INFINITA.

- C SE EJECUTÓ 5 VECES
- A DOS VECES
- B SOLAMENTE UNA
- EXACTAMENTE EN PROPORCIÓN A SUS VALORES DE BOLETOS DE 250, 100 Y 50, RESPECTIVAMENTE.
- LA PLANIFICACIÓN DE ZANCADAS LOGRA LAS PROPORCIONES DE FORMA EXACTA

	Pass(A) (stride=100)	Pass(B) (stride=200)	Pass(C) (stride=40)	Who Runs?
	0	0	0	A
	100	0	0	B
	100	200	0	C
	100	200	40	C
	100	200	80	C
	100	200	120	A
	200	200	120	C
	200	200	160	C
	200	200	200	...

- DADA LA PRECISIÓN DE LA PLANIFICACIÓN DE ZANCADAS, ¿POR QUÉ UTILIZAR LA PLANIFICACIÓN DE LOTERÍA?
- NO ES NECESARIO TOMAR EN CUENTA UN **ESTADO GLOBAL**.
 - UN NUEVO PROCESO ENTRA EN LA MITAD DEL EJEMPLO ANTERIOR: ¿CUÁL DEBE SER SU VALOR DE PASOS?
 - ¿DEBERÍA SER CERO? MONOPOLIZARÍA EL CPU
 - CON LOTERÍA, SIMPLEMENTE SE AGREGA UN NUEVO PROCESO A LA LISTA CON EL NÚMERO DE BOLETOS QUE TIENE, Y SE ACTUALIZA EL NÚMERO GLOBAL DE BOLETOS EXISTENTES.
- A VECES ES MÁS CONVENIENTE EVITAR EL MANTENIMIENTO DE ESTADO GLOBAL O UTILIZAR ALEATORIZACIÓN, A CAMBIO DE CIERTA PRECISIÓN.

LINUX COMPLETELY FAIR SCHEDULER

- IMPLEMENTA PLANIFICACIÓN DE PARTES JUSTAS DE UNA FORMA MUY EFICIENTE Y ESCALABLE.
- GASTA POCO TIEMPO REALIZANDO DECISIONES DE PLANIFICACIÓN, GRACIAS A:
 - DISEÑO
 - UTILIZACIÓN DE CIERTAS ESTRUCTURAS DE DATOS APROPIADAS
- LA EFICIENCIA DE LOS PLANIFICADORES ES SUMAMENTE IMPORTANTE
 - EN UN ESTUDIO REALIZADO EN LOS CENTROS DE DATOS DE GOOGLE, LA PLANIFICACIÓN REPRESENTÓ APROXIMADAMENTE EL 5% DE TODA LA UTILIZACIÓN DEL CPU.



OPERACIÓN BÁSICA

- TÉCNICA BASADA EN CONTEO
- CADA QUE UN PROCESO SE EJECUTA ACUMULA **VRUNTIME**
- EN EL CASO MÁS BÁSICO, EL VRUNTIME DE CADA PROCESO AUMENTA EN LA MISMA PROPORCIÓN ACORDE AL TIEMPO REAL.
- CUANDO UNA DECISIÓN DE PLANIFICACIÓN OCURRE, EL **CFS** ESCOGE EL PROCESO CON EL VRUNTIME MÁS BAJO

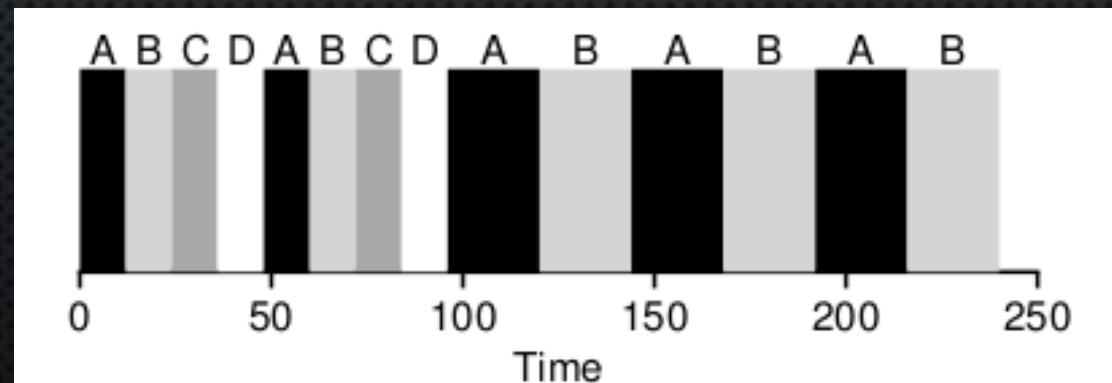
- SI SE CAMBIA DE PROCESOS MÁS PRONTO, EL ALGORITMO SERÁ MÁS JUSTO.
- CADA PROCESO RECIBIRÁ SU PARTE DEL CPU EN PARTES MUY PEQUEÑAS DE TIEMPO, A COSTA DE UN RENDIMIENTO DISMINUIDO (MUCHOS CAMBIOS DE CONTEXTO).
- SI SE EFECTÚAN POCOS CAMBIOS DE CONTEXTO AUMENTARÁ EL RENDIMIENTO, PERO SERÁ MENOS JUSTO.
- SE DEBE ENCONTRAR UN BALANCE. El CFS MANEJA ESTA SITUACIÓN MEDIANTE DIFERENTES PARÁMETROS.

SCCHED_LATENCY

- DETERMINA CUÁNTO TIEMPO SE DEBEN EJECUTAR LOS PROCESOS ANTES DE CONSIDERAR UN CAMBIO.
- UN VALOR TÍPICO ES DE 48 MILISEGUNDOS
- CFS DIVIDE ESTE VALOR ENTRE EL NÚMERO (N) DE PROCESOS EJECUTÁNDOSE EN EL CPU PARA DETERMINAR LA PARTE DE TIEMPO PARA UN PROCESO.
- EN ESTE PERÍODO DE TIEMPO, EL ALGORITMO SERÁ JUSTO.

EJEMPLO

- $N = 4$ PROCESOS EJECUTÁNDOSE
- SE DIVIDE EL VALOR DE SCHED_LATENCY ENTRE N, RESULTANDO EN PEDAZOS DE 12 MILISEGUNDOS POR PROCESO.
- CFS PLANIFICA EL PRIMER PROCESO Y LO EJECUTA HASTA QUE UTILIZA 12 MILISEGUNDOS DE TIEMPO DE EJECUCIÓN VIRTUAL (VIRTUAL RUNTIME)
- SE COMPRUEBA SI HAY UN PROCESO CON MENOR VRUNTIME. DE SER ASÍ, SE EJECUTA, Y ASÍ SUCEΣIVAMENTE.



MIN_GRANULARITY

- ¿QUÉ PASA SI HAY MUCHOS PROCESOS EJECUTÁNDOSE AL MISMO TIEMPO?
 - LOS PEDAZOS DE TIEMPO SERÍAN MUY PEQUEÑOS
 - SE TENDRÍAN QUE HACER MUCHOS CAMBIOS DE CONTEXTO.
- PARA RESOLVERLO, SE AGREGA LA GRANULARIDAD MÍNIMA, CON UN VALOR CERCANO A 6 MS USUALMENTE.
- CFS NUNCA LE ASIGNARÁ UN VALOR MENOR A ÉSTE A UN PEDAZO DE TIEMPO.
- DE ESTA FORMA, NO SE GASTA MUCHO TIEMPO EN EL OVERHEAD DE PLANIFICACIÓN Y SIGUE SIENDO JUSTO DE FORMA APROXIMADA.

EJEMPLO

- 10 PROCESOS EJECUTÁNDOSE
- EL CÁLCULO ORIGINAL RESULTARÍA EN PEDAZOS DE 4.8 MS.
- GRACIAS A MIN_GRANULARITY, CFS ASIGNARÍA LAS PARTES DE TIEMPO A 6 MS EN LUGAR DE 4.8.
- CFS NO SERÁ PERFECTAMENTE JUSTO DURANTE EL PERÍODO DE SCHED_LATENCY, PERO SE ENCONTRARÁ CERCANO MIENTRAS MANTIENE BUENA EFICIENCIA.

PONDERACIÓN (NICENESS)



- CFS PERMITE CIERTO CONTROL SOBRE LA PRIORIDAD DE LOS PROCESOS, PERMITIENDO A LOS USUARIOS O ADMINISTRADORES DARLES UN PORCENTAJE MAYOR DEL CPU.
- NO SE REALIZA MEDIANTE BOLETOS, SINO MEDIANTE UN MECANISMO DE UNIX CONOCIDO COMO EL NIVEL **NICE** DE UN PROCESO.
- PUEDE TOMAR VALORES DESDE -20 HASTA $+19$, CON UN VALOR POR DEFAULT DE 0 .
- VALORES POSITIVOS IMPLICAN MENOR PRIORIDAD Y LOS NEGATIVOS IMPLICAN MAYOR PRIORIDAD.
- CUANDO SE ES MUY BUENO (**NICE**), NO SE OBTIENE MUCHA ATENCIÓN (DE PLANIFICACIÓN).

CFS MAPEA LOS VALORES NICE DE CADA PROCESO A CIERTO PESO, COMO SE MUESTRA A CONTINUACIÓN:

```
static const int prio_to_weight[40] = {  
    /* -20 */ 88761, 71755, 56483, 46273, 36291,  
    /* -15 */ 29154, 23254, 18705, 14949, 11916,  
    /* -10 */ 9548, 7620, 6100, 4904, 3906,  
    /* -5 */ 3121, 2501, 1991, 1586, 1277,  
    /* 0 */ 1024, 820, 655, 526, 423,  
    /* 5 */ 335, 272, 215, 172, 137,  
    /* 10 */ 110, 87, 70, 56, 45,  
    /* 15 */ 36, 29, 23, 18, 15,  
};
```

- ESTOS PESOS NOS PERMITEN CALCULAR LAS PARTES DE TIEMPO EFECTIVAS DE CADA PROCESO (DE LA MISMA FORMA EN QUE SE REALIZÓ ANTERIORMENTE), PERO AHORA TOMANDO EN CUENTA SUS DIFERENCIAS EN PRIORIDAD.
- LA FÓRMULA UTILIZADA PARA REALIZAR LO ANTERIOR ES LA SIGUIENTE:

$$\text{time_slice}_k = \frac{\text{weight}_k}{\sum_{i=0}^{n-1} \text{weight}_i} \cdot \text{sched_latency}$$

EJEMPLO

- SE TIENEN DOS PROCESOS: A Y B.
- A ES UN PROCESO IMPORTANTE, SE LE DA UN VALOR NICE -5.
- B NO ES TAN IMPORTANTE, TIENE UN VALOR NICE POR DEFAULT DE 0.
- SEGÚN LA TABLA DE PESOS, EL PESO DE A SERÁ 3121 Y EL DE B SERÁ 1024.
- AL PROCESO A LE CORRESPONDE APROXIMADAMENTE $\frac{3}{4}$ DE SCHED_LATENCY (ES DECIR, 36 MS), MIENTRAS QUE A B APROXIMADAMENTE $\frac{1}{4}$ DE SCHED_LATENCY (ES DECIR, 12 MS).

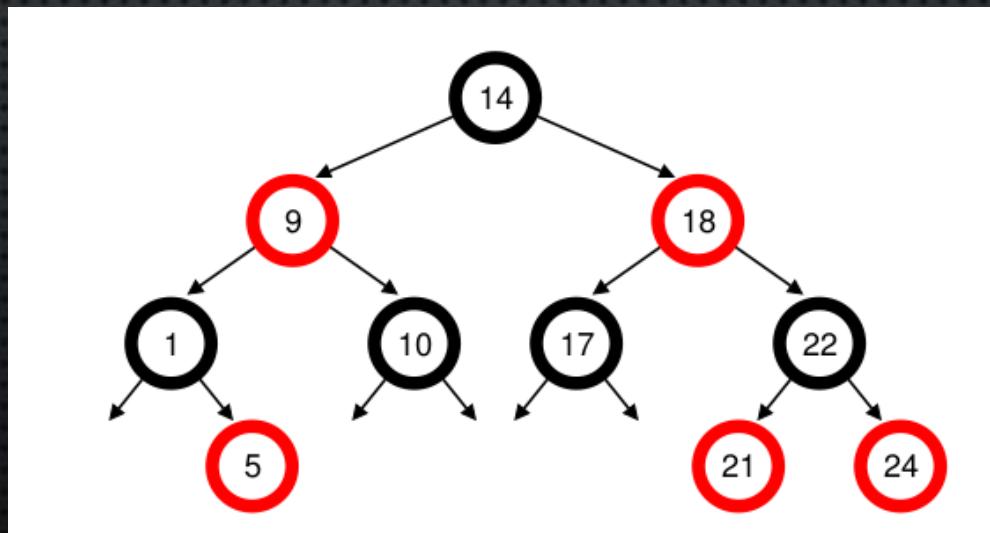
- LA FORMA EN LA QUE SE CALCULA EL VRUNTIME TAMBIÉN DEBE SER ADAPTADA.
- LA NUEVA FÓRMULA TOMA EN CUENTA EL TIEMPO DE EJECUCIÓN QUE EL PROCESO HA ACUMULADO, Y LO ESCALA INVERSAMENTE POR EL PESO DEL PROCESO.
- EN EL EJEMPLO ANTERIOR, EL TIEMPO DE EJECUCIÓN DE A SE ACUMULARÁ A UNA RAZÓN DE 1/3 CON RESPECTO AL TIEMPO ACUMULADO DE B.

$$\text{vruntime}_i = \text{vruntime}_i + \frac{\text{weight}_0}{\text{weight}_i} \cdot \text{runtime}_i$$

UTILIZANDO ÁRBOLES ROJINEGROS

- CFS DEBE SER EFICIENTE.
- ¿CÓMO ENCONTRAR EL SIGUIENTE PROCESO A EJECUTAR, CON EL VRUNTIME MÁS BAJO?
- ESTRUCTURAS SIMPLES COMO LISTAS NO SON ESCALABLES A CARGAS GRANDES. LA INSERCIÓN A UNA LISTA ORDENADA SE REALIZA EN $O(n)$.
- CFS ALMACENA LOS PROCESOS EN EJECUCIÓN EN UN ÁRBOL ROJINEGRO

- ES UN ÁRBOL BINARIO AUTOBALANCEADO. ES DECIR, A DIFERENCIA DE UN ÁRBOL BINARIO SIMPLE QUE SE PUEDE DEGENERAR HASTA TENER ALTURAS DE $O(n)$, ÉSTE SIEMPRE TENDRÁ ALTURAS DE $O(\log n)$
- GARANTIZA QUE LAS OPERACIONES A REALIZAR (COMO INSERCIÓN, BÚSQUEDA, O ELIMINACIÓN) SEAN DE TIEMPO LOGARÍTMICO.
- CFS SOLO MANTIENE EN ESTA ESTRUCTURA PROCESOS EN EJECUCIÓN O LISTOS PARA EJECUTAR. SI UN PROCESO “DUERME”, SE REMOVERÁ DEL ÁRBOL Y SE MANTENDRÁ EN OTRA PARTE DEL PLANIFICADOR.



TRATANDO CON ENTRADA/SALIDA Y PROCESOS DURMIENTES

- ¿QUÉ PASA SI HAY PROCESOS QUE DUERMEN POR PERÍODOS LARGOS DE TIEMPO?
- SE TIENEN DOS PROCESOS: A Y B
- A SE EJECUTA DE FORMA CONTINUA MIENTRAS QUE B DUERME POR 10 SEGUNDOS.
- CUANDO B DESPIERTA, SU VRUNTIME ESTARÁ 10 SEGUNDOS POR DEBAJO DEL DE A
- B AHORA MONOPOLIZARÁ EL CPU POR LOS DIEZ SEGUNDOS SIGUIENTES HASTA QUE SU VRUNTIME IGUALÉ AL DE A.

- CFS RESUELVE ESTE PROBLEMA AL ALTERAR EL VRUNTIME DE UN PROCESO CUANDO DESPIERTA.
- LE ASIGNA UN NUEVO VALOR DE VRUNTIME IGUAL AL VALOR MÍNIMO ENCONTRADO EN EL ÁRBOL.
- DE ESTA FORMA, SE EVITA LA INANICIÓN, PERO A CIERTO COSTO...
 - LOS PROCESOS QUE DUERMEN POR PERIODOS DE TIEMPO CORTOS DE FORMA FRECUENTE NO OBTIENEN SU PARTE JUSTA DEL CPU.



MÁS FUNCIONALIDADES DEL CFS

- NUMEROSES HEURÍSTICAS PARA MEJORAR EL RENDIMIENTO DEL CACHÉ.
- ESTRATEGIAS PARA MANEJAR MÚLTIPLES PROCESADORES DE FORMA EFECTIVA.
- PUEDE PLANIFICAR LARGOS GRUPOS DE PROCESOS (EN VEZ DE TRATAR A CADA PROCESO COMO UNA ENTIDAD INDIVIDUAL).
- Y MUCHAS OTRAS CARACTERÍSTICAS INTERESANTES...
- LO PRESENTADO ES ÚNICAMENTE UNA INTRODUCCIÓN. MÁS INFORMACIÓN SE ENCUENTRA DISPONIBLE EN LA SECCIÓN DE FUENTES CONSULTADAS.

RESUMEN

- PLANIFICACIÓN DE PARTES JUSTAS O PROPORCIONALES. TRES ENFOQUES:
 - PLANIFICACIÓN DE LOTERÍA, UTILIZA LA ALEATORIZACIÓN PARA LOGRAR “REPARTIR” EL CPU EN PARTES PROPORCIONALES DE FORMA APROXIMADA.
 - ZANCADAS LOGRA LO MISMO, PERO DE FORMA DETERMINÍSTICA.
 - CFS PUEDE RESUMIRSE COMO UN ROUND-ROBIN CON PESOS Y PARTES DE TIEMPO DINÁMICAS, CONSTRUIDO PARA ESCALAR Y RENDIR BIEN CON CARGAS GRANDES DE TRABAJO.

- NINGÚN PLANIFICADOR ES ÓPTIMO PARA TODOS LOS CASOS DE USO.
- LOS PLANIFICADORES DE PARTES JUSTAS TAMBIÉN TIENEN PROBLEMAS.
 - NO FUNCIONAN DE FORMA EXCELENTE CON LA ENTRADA/SALIDA.
 - ALGUNOS NO SON RELEVANTES EN CIERTOS DOMINIOS. POR EJEMPLO, EN EL CASO DE UN CENTRO DE DATOS VIRTUALIZADO, SE PODRÍA DESEAR ASIGNAR CIERTA CANTIDAD DE LOS CICLOS DE CPU A UNA MÁQUINA VIRTUAL.

FUENTES CONSULTADAS

- REMZI H. ARPACI-DUSSEAU, ANDREA C. ARPACI-DUSSEAU. (2008). SCHEDULING: PROPORTIONAL SHARE. EN OPERATING SYSTEMS: THREE EASY PIECES (743). WISCONSIN, ESTADOS UNIDOS: ARPACI-DUSSEAU BOOKS, LLC.
- LOTTERY SCHEDULING: FLEXIBLE PROPORTIONAL-SHARE RESOURCE MANAGEMENT, POR CARL A. WALDSPURGER Y WILLIAM E. WEHL. OSDI '94, NOVIEMBRE 1994.
- INSIDE THE LINUX 2.6 COMPLETELY FAIR SCHEDULER, POR M. TIM JONES. DICIEMBRE 15, 2009.
- LOTTERY AND STRIDE SCHEDULING: FLEXIBLE PROPORTIONAL-SHARE RESOURCE MANAGEMENT, POR CARL A. WALDSPURGER. TESIS DE DOCTORADO, MIT, 1995.