MODUL SISTEM OPERASI

(CCI210)

MODUL SESI 5

SCHEDULING

DISUSUN OLEH

ADI WIDIANTONO, SKOM, MKOM.

UNIVERSITAS ESA UNGGUL

2020

SCHEDULING (PENJADWALAN)

A. Kemampuan Akhir Yang Diharapkan

Setelah mempelajari modul ini, diharapkan mahasiswa mampu :

1. Memahami konsep dasar dari scheduling dari proses.
2. Memahami algoritma scheduling.
3. Mamahami proses kerja Sistem Operasi dalam menjalankan scheduling

B. Uraian dan Contoh

1. Scheduling/Penjadwalan

Kembali ke masa lalu sistem batch dengan masukan dalam bentuk gambar aktif kartu diatas pita magnetik, algoritme penjadwalannya sederhana: jalankan saja pekerjaan berikutnya yang ada di tape. Dengan sistem multiprogramming, algoritma penjadwalan menjadi lebih rumit karena bebepa pengguna yang menunggu layanan. Beberapa mainframe masih menggabungkan layanan batch dan timesharing, membutuhkan penjadwal/scheduler untuk memutuskan apakah pekerjaan batch atau pengguna interaktif di terminal harus dilanjutkan. (Selain itu, pekerjaan batch mungkin merupakan permintaan untuk menjalankan beberapa program secara berurutan, hal dapat dianggap permintaan untuk menjalankan satu program.) Karena waktu CPU adalah sumber daya yang langka pada mesin ini, penjadwal yang baik dapat membuat perbedaan besar dalam kinerja yang dirasakan dan kepuasan pengguna. Akibatnya, telah banyak dilakukan untuk merancang algoritme penjadwalan yang cerdas dan efisien.

Ketika kita beralih ke server jaringan, situasinya berubah secara signifikan. Disini banyak proses bersaing untuk mendapatkan CPU, jadi penjadwalan merupakan hal yang penting. Misalnya, ketika CPU harus memilih antara menjalankan proses yang mengumpulkan file statistik harian atau melayani permintaan pengguna, pengguna akan jauh lebih senang jika yang terakhir mendapat celah pertama di CPU.

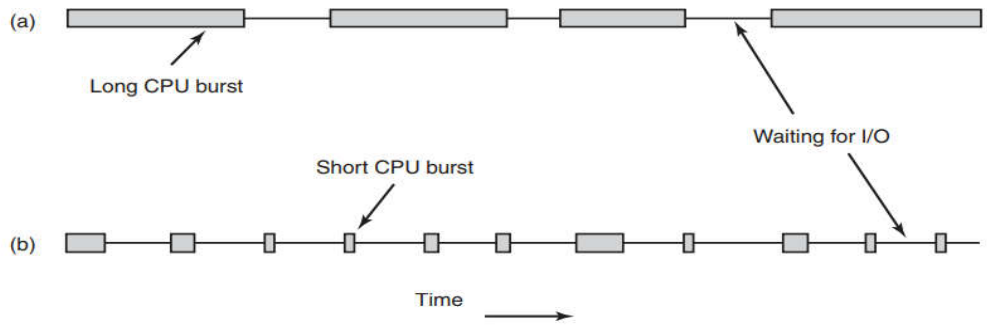
Argumen "sumber daya berlimpah/*abudance of resource* " tidak berlaku pada banyak perangkat mobile, seperti ponsel cerdas dan node dalam jaringan sensor. Selian CPU mungkin masih lemah dan memori kecil. masa pakai baterai adalah

salah satu kendala terpenting dari perangkat, beberapa system penjadwal mencoba mengoptimalkan konsumsi daya.

Selain memilih proses yang tepat untuk dijalankan, scheduler juga harus memperhatikan tentang penggunaan CPU yang efisien karena proses switching/peralihan mahal. Untuk mulai dengan, peralihan dari mode pengguna ke mode kernel, kemudian status proses saat ini harus disimpan, termasuk menyimpan registernya di tabel proses sehingga dapat dimuat ulang nanti. Di beberapa sistem, peta memori, harus disimpan juga. Selanjutnya proses baru harus dilakukan dan dipilih dengan menjalankan algoritma penjadwalan. Setelah itu, manajemen memori unit (MMU) harus dimuat ulang dengan peta memori dari proses baru. Akhirnya, proses baru harus dimulai. Secara keseluruhan, melakukan terlalu banyak swap/switch proses per detik dapat menghabiskan banyak waktu CPU.

1.1. Prilaku Proses (*Process Behavior*)

Hampir semua proses *burst of computation/CPU burst* bergantian dengan (disk atau jaringan) menunggu permintaan I/O, seperti yang ditunjukkan pada Gambar 1.1. Seringkali, CPU berjalan sebentar tanpa henti, kemudian panggilan sistem (*system call*) dibuat untuk membaca dari file atau menulis ke file. Saat system panggilan selesai, CPU menghitung lagi hingga membutuhkan lebih banyak data atau harus menulis lebih banyak data, dan seterusnya. Ada beberapa aktivitas I/O dihitung sebagai komputasi. Misalnya, ketika CPU menyalin bit ke RAM video untuk memperbarui layar, dianggap proses komputasi, bukan I/O, karena CPU sedang digunakan. I/O dalam pengertian ini adalah ketika suatu proses memasuki keadaan diblokir menunggu perangkat eksternal menyelesaikan pekerjaannya.



Gambar 1.1. Bursts of CPU usage alternate with periods of waiting for I/O. (a) A CPU-bound process. (b) An I/O-bound process.

Compute-Bound/CPU-Bound

Dalam Gambar 1.1.(a) proses menghabiskan sebagian besar waktu untuk komputasi, disebut compute-bound atau CPU-bound. Proses yang terikat komputasi biasanya memiliki CPU burst yang lama sehingga jarang menunggu I/O.

I/O-Bound

Sementara proses (b), menghabiskan sebagian besar waktu untuk menunggu I/O, disebut I/O bound. Umumnya proses yang terikat I / O memiliki CPU burst yang singkat sehingga sering menunggu I/O. Faktor lamanya CPU burst, bukan dari panjang bust I / O. Proses yang I/O-Bound terkait dengan I/O karena tidak menghitung banyak antara permintaan I/O, bukan karena permintaan I/O mereka sangat panjang. Dibutuhkan waktu yang sama untuk mengeluarkan permintaan perangkat keras untuk membaca blok disk tidak peduli seberapa banyak waktu yang diperlukan untuk memproses data setelah data tersebut tiba.

Perlu dicatat bahwa karena CPU semakin cepat, proses cenderung mendapatkan lebih banyak I/Obound. Efek ini terjadi karena CPU meningkat jauh lebih cepat daripada disk. Sebagai akibatnya, penjadwalan proses terikat I/O-bound cenderung menjadi subjek lebih penting di masa depan. Ide dasarnya di sini adalah jika proses I/O-bound ingin dijalankan, ia harus mendapatkan kesempatan dengan cepat sehingga ia dapat mengeluarkan permintaan disk dan biarkan disk sibuk.

1.2. Kapan membutuhkan penjadwalan

Masalah utama terkait penjadwalan adalah kapan harus membuat keputusan penjadwalan. Ada berbagai situasi yang membutuhkan penjadwalan:

Pertama, ketika proses baru dibuat, keputusan perlu dibuat apakah akan menjalankan proses induk atau proses anak. Karena kedua proses dalam keadaan siap, ini adalah keputusan penjadwalan normal dan dapat berjalan dengan cara apa pun, yaitu, penjadwal dapat secara sah memilih untuk menjalankan induk atau turunannya.

Kedua, keputusan penjadwalan harus dibuat saat proses keluar. Proses itu tidak bisa lagi berjalan (karena sudah tidak ada lagi), jadi beberapa proses lain harus dijalankan dipilih dari rangkaian proses yang siap. Jika tidak ada proses yang siap, proses idle biasanya dijalankan.

Ketiga, ketika sebuah proses memblokir I / O, pada semaphore, atau karena alasan lain, proses lain harus dipilih untuk dijalankan. Terkadang alasan pemblokiran mungkin memainkan peran dalam pemilihan. Misalnya, jika A adalah proses yang penting dan memang demikian menunggu B untuk keluar dari wilayah kritisnya, membiarkan B berjalan selanjutnya akan memungkinkannya keluar wilayah kritis dan dengan demikian biarkan A melanjutkan. Masalahnya adalah penjadwal umumnya tidak memiliki informasi yang diperlukan untuk memperhitungkan ketergantungan ini.
Keempat, ketika terjadi interupsi I/O, keputusan penjadwalan dapat dibuat. Jika interupsi berasal dari perangkat I/O yang sekarang telah menyelesaikan pekerjaannya, beberapa proses yang diblokir menunggu I/O sekarang mungkin siap untuk dijalankan. Terserah kepada penjadwal untuk memutuskan apakah akan menjalankan proses yang baru siap, atau proses itu berjalan pada saat interupsi, atau beberapa proses ketiga.

Jika clock perangkat keras memberikan interupsi berkala pada 50 atau 60 Hz atau frekuensi lainnya, keputusan penjadwalan dapat dibuat di setiap interupsi clock atau di setiap interupsi clock ke-k. Algoritma penjadwalan dapat dibagi menjadi dua kategori dengan memperhatikan bagaimana menangani interupsi clock.

Algoritme penjadwalan **nonpreemptive** memilih sebuah proses untuk dijalankan dan kemudian membiarkannya berjalan sampai diblokir (baik pada I/O atau menunggu proses lain) atau melepaskan CPU secara sukarela. Bahkan jika berjalan berjam-jam, tetap tidak akan ditangguhkan secara paksa. Akibatnya, tidak ada keputusan penjadwalan dibuat selama interupsi clock. Setelah pemrosesan interupsi clock selesai, proses yang berjalan sebelum interupsi dilanjutkan, kecuali ada proses dengan prioritas yang lebih tinggi sedang menunggu batas waktu yang sekarang terpenuhi.
Sebaliknya, algoritma penjadwalan **preemptive** memilih sebuah proses dan membiarkannya berjalan selama maksimum beberapa waktu yang tetap. Jika masih berjalan di akhir interval waktu, maka ditangguhkan dan penjadwal memilih proses lain untuk dijalankan (jika tersedia). Melakukan penjadwalan preemptive memerlukan clock interupsi terjadi di akhir interval waktu untuk memberikan kendali atas CPU kembali ke penjadwal. Jika tidak tersedia clock, penjadwalan nonpreemptive adalah satu-satunya pilihan.

1.3. Kategori Algoritma Penjadwalan

Tiga kategori algoritma penjadwalan berdasrkan lingkungannya :

1. Batch
2. Interactive
3. Real Time

Sistem batch masih digunakan secara luas di dunia bisnis untuk melakukan penggajian, persediaan, piutang, hutang, perhitungan bunga (di bank), pemrosesan klaim (di perusahaan asuransi), dan tugas berkala lainnya. Dalam sistem batch, tidak ada pengguna yang tidak sabar menunggu di terminal mereka untuk mendapatkan respons cepat atau untuk permintaan singkat. Akibatnya, algoritma nonpreemptive, atau algoritma preemptive dengan periode waktu yang lama untuk setiap proses, seringkali dapat diterima. Pendekatan ini mengurangi switch proses dan dengan demikian meningkatkan kinerja.

Dalam lingkungan dengan pengguna interaktif, preemption penting untuk mempertahankan proses dari memonopoli CPU dan menolak layanan ke yang lain. Bahkan jika tidak ada proses yang dengan sengaja berjalan selamanya, satu proses mungkin menutup semua proses lainnya tanpa batas karena bug program. Preemption diperlukan untuk mencegah perilaku ini. Server juga termasuk dalam kategori ini, karena mereka biasanya melayani beberapa pengguna (jarak jauh), semuanya yang sedang terburu-buru.

Dalam sistem dengan waktu nyata (real time), preemption, terkadang tidak diperlukan karena proses tahu bahwa mungkin tidak berjalan untuk waktu yang lama dan biasanya melakukan pekerjaan dan memblokir dengan cepat. Bedanya dengan interaktif sistem adalah bahwa sistem real-time hanya menjalankan program yang sudah ditangan. Sistem interaktif merupakan penggunaan umum dan program dapat berjalan secara sewenang-wenang yang mungkin tidak kooperatif dan bahkan berbahaya.

1.4. Scheduling Algorithm Goals

Beberapa goals (tujuan) dari algotima scheduling dapat dilihat sbb:

All systems

Fairness - giving each process a fair share of the CPU

Policy enforcement - seeing that stated policy is carried out

Balance - keeping all parts of the system busy

Batch systems

Throughput - maximize jobs per hour

Turnaround time - minimize time between submission and termination

CPU utilization - keep the CPU busy all the time

Interactive systems

Response time - respond to requests quickly

Propor tionality - meet users' expectations

Real-time systems

Meeting deadlines - avoid losing data

Predictability - avoid quality degradation in multimedia systems

2. Penjadwalan dalam system Batch

Algoritma penjadwalan yang digunakan dalam sistem Batch:

First Come First Serve (FCFS)

Mungkin yang paling sederhana dari semua algoritme penjadwalan yang pernah dibuat adalah nonpreemptive first-come, first-serve (FCFS). Dengan algoritma ini, proses diberi CPU dalam urutan yang diminta. Pada dasarnya, ada satu antrian proses siap. Ketika pekerjaan pertama memasuki sistem dari luar di awal, akan dimulai segera dan berjalan selama yang diinginkan (sampai selesai) dan tidak akan terputus walaupun sudah berjalan terlalu lama. Saat pekerjaan lain masuk, mereka ditempatkan di ujung antrian. Ketika proses yang sedang berjalan terjadi blokir, proses pertama pada antrian dijalankan selanjutnya. Ketika proses yang diblokir menjadi siap, diperlakukan seperti pekerjaan yang baru tiba, diletakkan di akhir antrian, di belakang semua proses yang menunggu.

FCFS memiliki kelemahan yang cukup besar. Misalkan ada satu proses CPU-Bound yang berjalan selama 1 detik pada satu waktu dan banyak lagi proses I/O-bound yang menggunakan sedikit waktu CPU tetapi masing-masing harus menjalankan

membaca 1000 disk sampai selesai. Proses CPU-bound berjalan selama 1 detik, lalu membaca disk blok. Semua proses I/O sekarang berjalan dan mulai membaca disk. Saat proses CPU-bound mendapatkan blok disknya, proses itu berjalan selama 1 detik lagi, diikuti oleh semua proses I/O-bound secara berurutan. Hasil akhirnya adalah bahwa setiap proses yang I/O-bound mendapat pembacaan 1 blok per detik dan membutuhkan waktu 1000 detik untuk menyelesaikannya. Dengan algoritme penjadwalan yang preeempted proses CPU-bound setiap 10 msec, proses I/O-bound akan selesai dalam 10 detik, bukannya 1000 detik, dan tanpa memperlambat banyak proses CPU-bound.

Shortest Job First (SJF)

Algoritma batch nonpreemptive lain yang mengasumsikan waktu proses diketahui sebelumnya. Ketika beberapa proses yang sama pentingnya sedang berlangsung antrian masukan menunggu untuk dimulai, penjadwal memilih proses dengan waktu terpendek terlebih dahulu.

Gambar 2.1.

Lihat Gambar 2.1. ada empat pekerjaan A, B, C, dan D dengan waktu berjalan 8, 4, 4, dan 4 menit. Dengan menjalankannya dalam urutan itu, waktu penyelesaian untuk A adalah 8 menit, untuk B adalah 12 menit, untuk C adalah 16 menit, dan untuk D adalah 20 menit untuk rata-rata 14 menit. Pertimbangkan untuk menjalankan keempat pekerjaan ini menggunakan pekerjaan terpendek dulu, seperti yang ditunjukkan pada Gambar 2.1 (b). Waktu penyelesaiannya sekarang adalah 4, 8, 12, dan 20 menit selama rata-rata 11 menit. Pekerjaan terpendek dulu terbukti optimal. Pertimbangkan kasus empat pekerjaan, dengan waktu pelaksanaan a, b, c, dan d, masing-masing. Pekerjaan pertama selesai pada waktu a, detik, yang kedua pada waktu a + b, dan seterusnya. Waktu penyelesaian rata-rata adalah (4a + 3b + 2c + d) / 4. Jelas bahwa a memberikan kontribusi lebih pada rata-rata daripada yang lain, jadi harus menjadi pekerjaan terpendek, dengan b berikutnya, lalu c, dan terakhir d sebagai terpanjang karena hanya mempengaruhi waktu penyelesaiannya sendiri.

Argumen yang sama berlaku sama baiknya untuk sejumlah pekerjaan. Perlu ditunjukkan bahwa pekerjaan terpendek pertama akan optimal hanya jika semua pekerjaan tersedia secara bersamaan. Sebagai contoh, pertimbangkan lima pekerjaan, A sampai E, dengan run time masing-masing 2, 4, 1, 1, dan 1. Waktu kedatangan mereka adalah 0, 0, 3, 3, dan 3. Awalnya, hanya A atau B yang bisa dipilih, karena tiga job lainnya belum datang. Dengan menggunakan pekerjaan terpendek dulu, kita akan menjalankan pekerjaan dengan urutan A, B, C, D, E, untuk menunggu rata-rata 4,6. Namun, menjalankannya dalam urutan B, C, D, E, A memiliki rata-rata menunggu 4.4.

Shortest Remaining Time Next

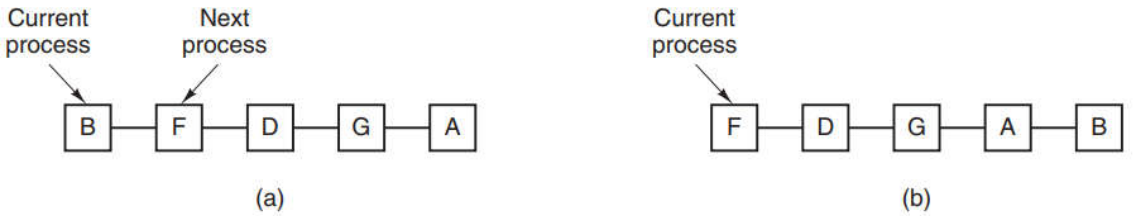
Versi preemptive dari SJF adalah waktu sisa terpendek berikutnya. Dengan algoritma ini, penjadwal selalu memilih proses yang tersisa run time adalah yang terpendek. Sekali lagi di sini, run time harus diketahui sebelumnya. Saat pekerjaan baru tiba, waktu totalnya dibandingkan dengan waktu proses saat ini yang tersisa. Jika pekerjaan baru membutuhkan lebih sedikit waktu untuk diselesaikan daripada proses saat ini, proses saat ini akan ditangguhkan dan pekerjaan baru dimulai. Skema ini memungkinkan urutan baru pekerjaan untuk mendapatkan layanan yang terbaik.

3. Penjadwalan dalam system Interaktif

Algoritma penjadwalan yang digunakan dalam system interaktif:

Round-Robin Scheduling

Salah satu algoritme tertua, paling sederhana, paling adil, dan paling banyak digunakan adalah algoritme round-robin. Setiap proses diberi interval waktu, yang disebut kuantum. Jika proses masih berjalan di akhir kuantum, CPU di-preemted dan diberikan ke proses lain. Jika proses telah diblokir atau diselesaikan sebelum kuantum berlalu, peralihan CPU dilakukan saat proses tersebut diblok. Round- robin mudah diimplementasikan. Semua penjadwal perlu lakukan adalah memelihara daftar proses yang dapat dijalankan, seperti yang ditunjukkan pada Gambar 2.2 (a). Ketika proses tersebut menggunakan kuantumnya, proses tersebut diletakkan di akhir daftar, seperti yang ditunjukkan pada Gambar 2.2 (b).



Gambar 2.2 round-robin

Satu-satunya masalah dengan round robin adalah panjang kuantum. Berpindah dari satu proses ke proses lainnya membutuhkan waktu tertentu untuk melakukannya semua administrasi — menyimpan dan memuat register dan peta memori, memperbarui berbagai tabel dan daftar, membersihkan dan memuat ulang cache memori, dan seterusnya. Misalkan proses ini atau saklar konteks, seperti yang kadang-kadang disebut, membutuhkan 1 msec, termasuk mengganti peta memori, membersihkan dan memuat ulang cache, dll. Misalkan kuantum diatur pada 4 msec. Dengan parameter tersebut, setelah melakukan 4 msec pekerjaan, CPU harus menghabiskan (mis., Membuang) 1 msec untuk proses beralih. Jadi, 20% dari waktu CPU akan dibuang untuk biaya administrasi.

Untuk meningkatkan efisiensi CPU, kita dapat mengatur kuantum menjadi, katakanlah, 100 msec. Sekarang waktu yang terbuang hanya 1%. Tetapi pertimbangkan apa yang terjadi pada sistem server jika 50 permintaan masuk dalam interval waktu yang sangat singkat dan dengan CPU yang memiliki kebutuhan sangat bervariasi. Lima puluh proses akan dimasukkan ke dalam daftar proses yang dapat dijalankan. Jika CPU menganggur, yang pertama akan segera dimulai, yang kedua mungkin tidak akan mulai sampai 100 msec kemudian, dan seterusnya. Yang terakhir tidak beruntung mungkin harus menunggu 5 detik sebelum mendapat kesempatan, dengan asumsi semua yang lain menggunakan kuantum penuh. Sebagian besar pengguna akan menganggap respons 5 detik untuk perintah singkat sebagai lamban. Situasi ini khususnya buruk jika beberapa permintaan mendekati akhir antrian hanya memerlukan beberapa milidetik waktu CPU. Dengan kuantum yang singkat, mereka akan mendapatkan layanan yang lebih baik.

Faktor lainnya adalah jika kuantum disetel lebih lama dari rata-rata ledakan CPU, preemption tidak akan sering terjadi. Sebaliknya, sebagian besar proses akan melakukan pemblokiran operasi sebelum kuantum habis, menyebabkan peralihan proses. Menghilangkan preemption meningkatkan kinerja karena proses beralih kemudian terjadi hanya jika secara logis diperlukan, yaitu, saat proses memblokir dan tidak bisa melanjutkan.

Kesimpulannya dapat dirumuskan sebagai berikut: pengaturan kuantum terlalu pendek menyebabkan terlalu banyak switch proses dan menurunkan efisiensi CPU, tetapi juga menyetelnya lama dapat menyebabkan respons yang buruk terhadap permintaan interaktif yang singkat. Sebuah kuantum di sekitar 20–50 msec sering kali merupakan kompromi yang masuk akal.

Priority Scheduling

Penjadwalan round-robin membuat asumsi implisit bahwa semua proses adalah sama pentingnya. Seringkali, orang-orang yang memiliki dan mengoperasikan komputer multiuser memiliki gagasan yang sangat berbeda tentang hal itu. Kebutuhan untuk mempertimbangkan faktor eksternal mengarah pada penjadwalan prioritas. Ide dasarnya sangat mudah: setiap proses diberi prioritas, dan proses yang dapat dijalankan dengan prioritas tertinggi diizinkan untuk dijalankan.

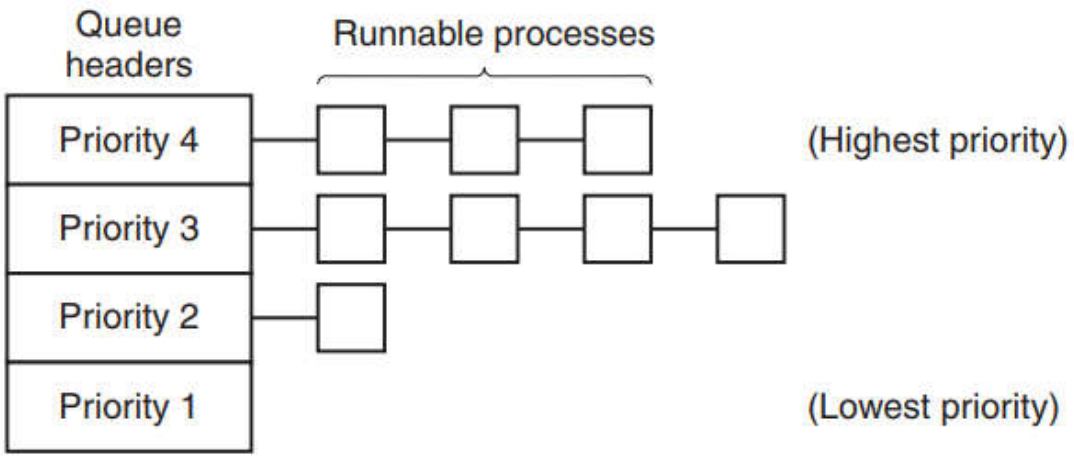
Bahkan pada PC dengan satu pemilik, mungkin ada beberapa proses, beberapa di antaranya mereka lebih penting dari yang lain. Misalnya, proses daemon yang mengirim surat elektronik di latar belakang harus diberi prioritas yang lebih rendah daripada proses menampilkan film video di layar secara real time.

Untuk mencegah proses prioritas tinggi berjalan tanpa batas, penjadwal dapat menurunkan prioritas dari proses yang sedang berjalan di setiap detak clock (yaitu, di setiap clock interupsi). Jika tindakan ini menyebabkan prioritasnya turun di bawah prioritas proses tertinggi berikutnya, terjadi peralihan proses. Atau, setiap proses mungkin menetapkan kuantum waktu maksimum yang diizinkan untuk dijalankan. Saat kuantum habis, proses dengan prioritas tertinggi berikutnya diberi kesempatan untuk dijalankan.

Prioritas juga dapat ditetapkan secara dinamis oleh sistem untuk mencapai tujuan tertentu. Misalnya, beberapa proses sangat I/O-bound dan menghabiskan sebagian besar waktu menunggu I/O selesai. Kapanpun proses seperti itu menginginkan CPU, itu harus diberikan CPU segera, untuk membiarkannya memulai permintaan I/O berikutnya, kemudian dapat melanjutkan secara paralel dengan proses lain yang merupakan proses *computing*. Algoritma sederhana untuk member layanan yang baik untuk proses yang terikat I/O-bound adalah menetapkan prioritas ke 1 / f, di mana f adalah pecahan dari kuantum terakhir yang digunakan proses. Sebuah proses yang hanya menggunakan 1 msec-nya Kuantum 50-msec akan mendapatkan prioritas 50, sedangkan proses yang sebelumnya berjalan 25 msec

pemblokiran akan mendapatkan prioritas 2, dan proses yang menggunakan seluruh kuantum akan mendapat prioritas 1.

Seringkali mudah untuk mengelompokkan proses ke dalam kelas prioritas dan menggunakan prioritas penjadwalan antar kelas daripada penjadwalan round-robin dalam setiap kelas. Angka 2-43 menunjukkan sistem dengan empat kelas prioritas. Algoritma penjadwalan adalah sebagai berikut: selama ada proses yang dapat dijalankan di kelas prioritas 4, jalankan saja masing-masing untuk satu gaya quantum, round-robin, dan tidak pernah peduli dengan kelas dengan prioritas lebih rendah. Jika kelas prioritas 4 kosong, maka jalankan proses kelas 3 round robin. Jika kelas 4 dan 3 keduanya kosong, lalu jalankan round robin kelas 2, dan seterusnya. Jika prioritas tidak disesuaikan sesekali, kelas dengan prioritas lebih rendah mungkin akan pernah berjalan.



Gambar 2.4. Priority Scheduling

Shortest Process Next

Karena pekerjaan terpendek dulu selalu menghasilkan waktu respons rata-rata minimum untuk sistem batch, alangkah baiknya jika dapat digunakan untuk proses interaktif juga. Proses interaktif umumnya mengikuti pola menunggu perintah, menjalankan perintah, menunggu perintah, menjalankan perintah, dll. Jika menganggap eksekusi setiap perintah sebagai " pekerjaan " yang terpisah, maka dapat meminimalkan waktu respons secara keseluruhan dengan menjalankan yang terpendek terlebih dahulu. Masalahnya adalah mencari tahu proses mana yang saat ini dapat dijalankan yang paling pendek.

Salah satu pendekatannya adalah membuat perkiraan berdasarkan perilaku masa lalu dan menjalankan prosesnya dengan perkiraan waktu berjalan terpendek. Misalkan perkiraan waktu per perintah untuk beberapa proses adalah T0. Sekarang misalkan proses berikutnya diukur menjadi T1. Memperbarui perkiraan dapat dengan mengambil jumlah tertimbang dari dua angka ini, yaitu, aT0 + (1 - a)T1. Melalui pilihan ini dapat memutuskan untuk memiliki estimasi proses dengan cepat, atau mengingatnya untuk waktu yang lama. Dengan a = 1/2, dapat perkiraan yang berurutan sbb:

T0, T0 / 2 + T1 / 2, T0 / 4 + T1 / 4 + T2 / 2, T0 / 8 + T1 / 8 + T2 / 4 + T3 / 2

Setelah tiga proses baru, bobot T0 di estimasi baru turun menjadi 1/8.

Teknik mengestimasi nilai selanjutnya secara berurutan dengan mengambil bobot rata-rata nilai terukur saat ini dan perkiraan sebelumnya kadang-kadang disebut ***aging***. Ini berlaku untuk banyak situasi di mana prediksi harus dibuat berdasarkan nilai sebelumnya. Penuaan sangat mudah diterapkan jika a = 1/2. Semua yang diperlukan adalah menambahkan nilai baru ke perkiraan saat ini dan membagi jumlahnya dengan 2 (dengan menggesernya ke kanan 1 bit).

Guaranteed Scheduling

Pendekatan yang sama sekali berbeda untuk penjadwalan adalah membuat janji nyata kepada pengguna tentang kinerja dan kemudian memenuhi janji tersebut. Satu janji itu realistis untuk membuat dan mudah untuk dipenuhi adalah: jika n pengguna login saat Anda berada berkerja, Anda akan menerima sekitar 1 / n dari daya CPU. Begitu pula pada satu pengguna sistem dengan n proses yang berjalan, semua hal dianggap sama, masing-masing harus mendapatkan 1 / n siklus CPU.

Untuk memenuhi janji ini, sistem harus melacak berapa banyak CPU setiap proses memiliki sejak pembuatannya. Kemudian menghitung jumlah CPU masing-masing yang berhak, yaitu waktu sejak penciptaan dibagi dengan n. Sejak jumlah waktu CPU yang dimiliki setiap proses juga diketahui, ini cukup mudah untuk menghitung rasio waktu CPU aktual yang dikonsumsi dengan waktu CPU yang berhak. Rasio 0,5 berarti bahwa suatu proses hanya memiliki setengah dari apa yang seharusnya dimilikinya, dan rasio 2.0 berarti bahwa suatu proses memiliki dua kali lipat dari yang seharusnya. Algoritma ini kemudian menjalankan proses dengan

rasio terendah sampai rasionya berpindah di atas pesaing terdekatnya. Kemudian yang itu dipilih untuk dijalankan berikutnya.

Lottery Scheduling

Meskipun membuat janji kepada pengguna dan kemudian menepati janji itu adalah ide yang bagus, namun sulit untuk diterapkan. Namun, algoritma lain dapat digunakan untuk memberi hasil yang serupa yang dapat diprediksi dengan implementasi yang lebih sederhana. Ini disebut lotere penjadwalan (Waldspurger dan Weihl, 1994).

Ide dasarnya adalah memberikan proses tiket lotere untuk berbagai sumber daya sistem, seperti waktu CPU. Setiap kali keputusan penjadwalan harus dibuat, tiket lotere dipilih secara acak, dan proses memegang tiket itu mendapatkan sumber daya. Kapan diterapkan pada penjadwalan CPU, sistem mungkin mengadakan lotere 50 kali per detik, dengan setiap pemenang mendapatkan 20 msec waktu CPU sebagai hadiah.

Untuk memparafrasekan George Orwell: “Semua proses adalah sama, tetapi beberapa proses lebih setara”. Proses yang lebih penting dapat diberikan tiket tambahan, untuk meningkatkan peluang mereka untuk menang. Jika ada 100 tiket yang beredar, dan satu proses ditangguhkan 20 di antaranya, akan memiliki peluang 20% untuk memenangkan setiap lotre. Dalam jangka panjang, itu akan mendapatkan sekitar 20% dari CPU. Berbeda dengan penjadwal prioritas, di mana itu sangat sulit untuk menyatakan apa sebenarnya arti memiliki prioritas 40, di sini aturannya jelas: sebuah proses memegang sebagian kecil dari tiket akan mendapatkan sekitar sebagian kecil dari sumber daya dalam pertanyaan.

Penjadwalan lotere memiliki beberapa sifat menarik. Misalnya, jika baru proses muncul dan diberikan beberapa tiket, pada undian berikutnya akan ada peluang menang sebanding dengan jumlah tiket yang dimilikinya. Dengan kata lain, penjadwalan lotere sangat responsif.

Proses kerja sama dapat bertukar tiket jika mereka mau. Misalnya, saat sebuah proses klien mengirim pesan ke proses server dan kemudian memblokir, mungkin memberikan semua tiketnya ke server, untuk meningkatkan kemungkinan server berjalan selanjutnya. Saat server selesai, ia mengembalikan tiket sehingga klien dapat berjalan kembali. Faktanya, jika klien tidak ada, server tidak membutuhkan tiket sama sekali.

Penjadwalan lotere dapat digunakan untuk menyelesaikan masalah yang sulit ditangani dengan metode lain. Salah satu contohnya adalah server video di mana beberapa proses berada memberikan streaming video ke klien mereka, tetapi pada frekuensi gambar yang berbeda. Misalkan file proses membutuhkan bingkai pada 10, 20, dan 25 bingkai / detik. Dengan mengalokasikan proses ini 10, 20, dan 25 tiket, masing-masing, mereka akan secara otomatis membagi CPU kira-kira proporsi yang benar, yaitu 10: 20: 25.

Fair-Share Scheduling

Jika diasumsikan bahwa setiap proses dijadwalkan sendiri-sendiri, tanpa memperhatikan siapa pemiliknya. Akibatnya, jika pengguna 1 memulai sembilan proses dan pengguna 2 memulai satu proses, dengan round robin atau prioritas yang sama, pengguna 1 akan mendapatkan 90% dari CPU dan pengguna 2 hanya 10% saja. Untuk mencegah situasi ini, beberapa sistem memperhitungkan pengguna mana yang memiliki sebuah proses sebelum menjadwalkannya. Dalam model ini, setiap pengguna dialokasikan beberapa bagian dari CPU dan penjadwal mengambil proses sedemikian rupa untuk memaksakannya. Jadi jika dua pengguna masing-masing telah dijanjikan 50% dari CPU, mereka masing-masing akan mendapatkannya, tidak peduli berapa banyak proses yang mereka miliki.

4. Penjadwalan dalam system Real Time

Sistem waktu nyata adalah sistem di mana waktu memainkan peran penting. Biasanya, satu atau lebih banyak perangkat fisik di luar komputer menghasilkan rangsangan, dan komputer harus bereaksi dengan tepat dalam jangka waktu tertentu. Sebagai contoh, komputer dalam pemutar compact disc mendapatkan bit saat mereka keluar dari drive dan harus mengubahnya menjadi musik dalam interval waktu yang sangat ketat. Jika dihitung memakan waktu terlalu lama, musik akan terdengar aneh. Sistem waktu nyata lainnya pemantauan pasien di unit perawatan intensif rumah sakit, autopilot di pesawat terbang, dan robot kontrol di pabrik otomatis. Dalam semua kasus ini, memiliki jawaban yang benar tetapi terlambat sering sama buruknya dengan tidak memilikinya sama sekali.

Sistem waktu nyata umumnya dikategorikan sebagai ***hard real time***, artinya adalah tenggat waktu mutlak yang harus dipenuhi — atau yang lain! - dan ***soft real time***, artinya bahwa melewatkan tenggat waktu sesekali tidak diinginkan, namun

tetap dapat ditoleransi. Di kedua kasus, perilaku real-time dicapai dengan membagi program menjadi angka proses, yang masing-masing perilakunya dapat diprediksi dan diketahui sebelumnya. Proses ini umumnya berumur pendek dan dapat berjalan sampai selesai dalam waktu kurang dari satu detik. Saat peristiwa eksternal terdeteksi, itu adalah tugas penjadwal untuk menjadwalkan proses sedemikian rupa sehingga semua tenggat waktu dipenuhi.

Peristiwa yang mungkin harus direspon oleh sistem real-time dapat dikategorikan lebih lanjut sebagai **periodik** (artinya terjadi pada interval reguler) atau **aperiodik** (artinya terjadi tidak terduga). Sebuah sistem mungkin harus merespon beberapa aliran periodik. Bergantung pada berapa banyak waktu yang dibutuhkan setiap acara untuk diproses,mmenangani semuanya bahkan mungkin tidak mungkin. Misalnya jika ada m periodic peristiwa dan peristiwa i terjadi dengan periode Pi dan membutuhkan Ci detik waktu CPU untuk menangani setiap peristiwa, maka beban hanya dapat ditangani jika

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

Sistem waktu nyata yang memenuhi kriteria ini disebut dapat dijadwalkan (scheduable). Ini berarti bisa diimplementasikan. Proses yang gagal untuk memenuhi pengujian ini tidak dapat dilakukan dijadwalkan karena jumlah total waktu CPU yang diinginkan proses secara kolektif lebih dari yang bisa dihasilkan CPU.

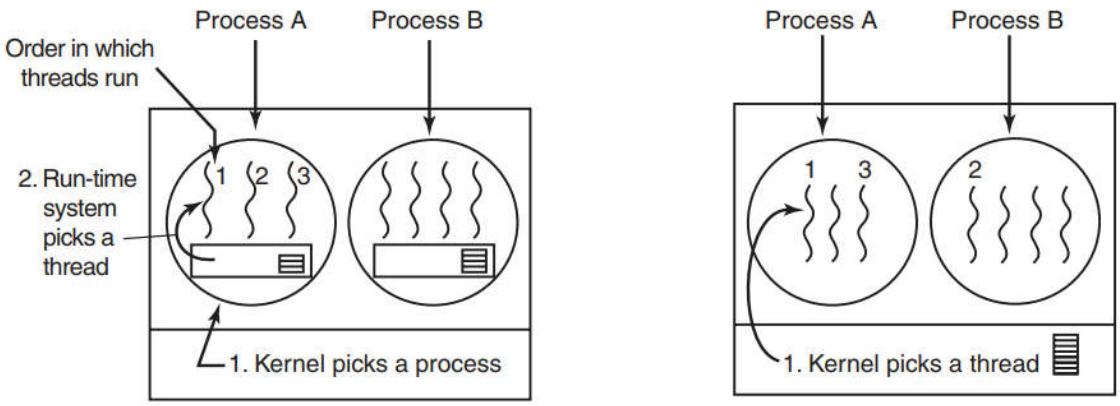
Sebagai contoh, pertimbangkan sistem real-time lunak dengan tiga kejadian periodik, dengan periode 100, 200, dan 500 mdet. Jika acara ini membutuhkan 50, 30, dan 100 msec waktu CPU per acara, masing-masing, sistem dapat dijadwalkan karena 0. 5 + 0. 15 + 0. 2 <1. Jika event keempat dengan periode 1 detik ditambahkan, maka sistem akan tetap dapat dijadwalkan selama acara ini tidak membutuhkan lebih dari 150 msec waktu CPU per acara. Yang tersirat dalam perhitungan ini adalah asumsi bahwa overhead pengalihan konteks sangat kecil sehingga dapat diabaikan.

Algoritme penjadwalan real-time dapat bersifat statis atau dinamis. Statis - membuat penjadwalan sebelum sistem mulai berjalan. Dinamis - membuat keputusan penjadwalan pada waktu proses, setelah eksekusi dimulai. Penjadwalan statis bekerja hanya jika ada informasi sempurna yang tersedia sebelumnya tentang pekerjaan tersebut yang harus diselesaikan dan tenggat waktu yang harus dipenuhi. Algoritma penjadwalan dinamis tidak memiliki batasan ini.

5. Thread Scheduling

Ketika beberapa proses masing-masing memiliki beberapa thread, ada dua tingkat paralelisme yaitu: proses dan thread. Penjadwalan dalam sistem tersebut sangat berbeda tergantung pada apakah **thread user level** atau **thread kernel level** (atau keduanya) didukung. Misalkan thread user level terlebih dahulu. Karena kernel tidak menyadari keberadaan thread, kernel beroperasi seperti biasa, memilih proses, katakanlah, A, dan memberikan A kontrol untuk kuantumnya. Penjadwal thread di dalam A memutuskan thread mana untuk menjalankan, katakan A1. Karena tidak ada interupsi clock multiprogram thread, ini thread dapat terus berjalan selama diinginkan. Jika itu menghabiskan seluruh proses kuantum, kernel akan memilih proses lain untuk dijalankan. Saat proses A akhirnya berjalan lagi, utas A1 akan kembali berjalan. Itu akan terus mengkonsumsi semua waktu A sampai selesai. Namun, perilaku antisosialnya tidak akan mempengaruhi proses lainnya. Mereka akan mendapatkan apa pun yang dianggap oleh penjadwal bagian yang sesuai, tidak peduli apa yang terjadi di dalam proses A.

Sekarang pertimbangkan kasus bahwa untaian A memiliki pekerjaan yang relatif sedikit untuk dilakukan per CPU burst, misalnya, pekerjaan 5 msec dalam kuantum 50-msec. Karena itu, masing-masing berjalan sebentar, lalu mengembalikan CPU ke penjadwal thread. Ini mungkin mengarah ke urutan A1, A2, A3, A1, A2, A3, A1, A2, A3, A1, sebelum kernel beralih ke proses B. Situasi ini diilustrasikan pada Gambar 5.1(a).



Gambar 5.1.

Algoritme penjadwalan yang digunakan oleh sistem run-time bisa salah satunya dijelaskan di atas. Dalam prakteknya, penjadwalan round-robin dan penjadwalan prioritas paling umum. Satu-satunya kendala adalah tidak adanya interupsi clock

untuk mengganggu thread yang sudah berjalan terlalu lama. Karena thread bekerja sama, hal ini biasanya tidak menjadi masalah.

Sekarang pertimbangkan situasi dengan Kernel level Thread. Di sini kernel memilih thread tertentu untuk dijalankan. Itu tidak harus memperhitungkan proses yang mana pemilik thread, tetapi dapat jika diinginkan. Thread diberi kuantum dan ditangguhkan secara paksa jika melebihi kuantum. Dengan kuantum 50-msec tetapi thread blok itu setelah 5 msec, urutan utas untuk beberapa periode 30 msec mungkin A1, B1, A2, B2, A3, B3, sesuatu yang tidak mungkin dilakukan dengan parameter dan user level thread. Situasi ini sebagian digambarkan pada Gambar 5.1. (b).

Perbedaan utama antara **user level thread** atau **kernel level thread** adalah kinerja. Melakukan peralihan utas dengan utas tingkat pengguna membutuhkan beberapa instruksi mesin. Dengan utas tingkat kernel, ini membutuhkan sakelar konteks penuh,
mengubah peta memori dan membatalkan cache, yang merupakan beberapa urutan besarnya lebih lambat. Di sisi lain, dengan utas tingkat kernel, memiliki utas blokir pada I / O tidak menangguhkan seluruh proses seperti halnya dengan utas tingkat pengguna.

Karena kernel mengetahui bahwa beralih dari utas dalam proses A ke utas masuk dari proses B lebih mahal daripada menjalankan utas kedua dalam proses A (karena harus mengubah peta memori dan cache memori rusak), ini dapat memakan waktu informasi ini diperhitungkan saat membuat keputusan. Misalnya, diberikan dua utas yang sebaliknya sama pentingnya, dengan salah satunya adalah milik proses yang sama seperti utas yang baru saja diblokir dan yang termasuk dalam proses berbeda, preferensi bisa diberikan kepada yang pertama.

Faktor penting lainnya adalah utas tingkat pengguna dapat menggunakan penjadwal utas khusus aplikasi. Misalkan thread pekerja baru saja diblokir dan thread petugas operator dan dua utas pekerja sudah siap. Siapa yang harus lari selanjutnya? Sistem run-time, tahu apa yang dilakukan semua utas, dapat dengan mudah memilih petugas operator untuk dijalankan berikutnya, sehingga bisa mulai pekerja lain berjalan. Strategi ini memaksimalkan jumlah paralelisme lingkungan di mana pekerja sering memblokir pada disk I / O. Dengan level kernel utas, kernel tidak akan pernah tahu apa yang dilakukan setiap utas (meskipun mereka bisa

menetapkan prioritas yang berbeda). Namun, secara umum, utas khusus aplikasi penjadwal dapat mengatur aplikasi lebih baik daripada kernel.

Daftar Pustaka

1. Modern Operating System 4th Ed. Andrew S Tanembaun 2009