

Review bab 15 "Penjadwalan Prosesor Jamak"

Anggota kelompok :

- Andri Satria (0606104196 - kelas A)
- Freddy Setiawan (0606104252 - kelas A)
- Hendy Kusumo Nugroho (0606101420 - kelas A)
- R. Brahmastro K (1205000746 – kelas B)

Komentar umum :

Menurut kami secara umum bab ini sudah cukup memadai dan mudah untuk dimengerti. jika dilihat berdasarkan buku Silberschatz bab "Penjadwalan Prosesor Jamak" pada buku MDGR ini sudah cukup bagus dan mencakup bagian-bagian penting dalam buku Silberschatz dan bahkan terdapat beberapa tambahan yang cukup bagus dan menambah pemahaman tentang prosesor jamak dan penjadwalannya.

Hubungan dengan bab sebelumnya dan selanjutnya :

Menurut hasil diskusi kami, bab "Penjadwalan Prosesor Jamak" kurang memiliki hubungan yang begitu kuat dengan bab sebelumnya yaitu "Algoritma Penjadwalan" dan bab sesudahnya yaitu "Evaluasi dan Ilustrasi". maksudnya disini adalah bab "Penjadwalan Prosesor Jamak" tidak menjelaskan lebih lanjut mengenai bab "Algoritma Penjadwalan", namun kurang lebih menjelaskan mengenai hal-hal yang perlu kita ketahui mengenai apa yang terjadi pada proses penjadwalan dalam prosesor jamak dan mekanismenya. Mungkin bab ini penting sebagai selingan antara bab sebelumnya dan sesudahnya karena selain penjadwalan pada single processor, kita perlu mengenal juga penjadwalan multi processor. Kemudian untuk bab "Evaluasi dan Ilustrasi" mungkin lebih berhubungan dan menjelaskan bab "Algoritma Penjadwalan" dan kurang ada hubungannya dengan bab "Penjadwalan Prosesor Jamak".

Komentar kelengkapan per bagian :

->bagian 15.1

- bagian ini sudah cukup bagus karena telah mencakup definisi singkat dari prosesor jamak, alasan yang membuat kita harus mempelajari mengenai prosesor jamak dan kelebihan dari prosesor jamak dibanding prosesor tunggal.

->bagian 15.2

- bagian ini kurang dilengkapi dengan contoh OS yang menggunakan penjadwalan Master/Slave.

- judulnya kurang sesuai terhadap bagian selanjutnya.

- gambar yang disajikan cukup membantu dalam pemahaman.

->bagian 15.3

- penjelasan yang diberikan sudah cukup untuk dimengerti.
- untuk memperjelas pemahaman dan agar lebih sesuai dengan bagian selanjutnya mungkin perlu ditambahkan ilustrasi gambar.
- mungkin perlu ditambahkan contoh tambahan lagi bila perlu untuk mengikuti perkembangan jaman.

->bagian 15.4

- kurang penjelasan mengenai soft affinity karena terjadi kontradiksi dengan definisi affinity. mungkin perlu diberi contoh.
- tidak dijelaskan definisi workload.

->bagian 15.5

- ada salah penulisan pada paragraf ke-2 line ke-7 terhadap kata chace.
- penjelasan yang diberikan sudah cukup baik dan sudah ada ilustrasi gambarnya.
- salah pada penulisan judul bagian 15.5

->bagian 15.6

- salah penulisan terhadap kata thread level paralelism.
- penjelasan sudah cukup bagus bahkan juga diterangkan mengenai keuntungan dan kerugian multi core.

->bagian 15.7

- mungkin perlu ditambahkan definisi singkat multi processor.
- rangkuman sudah mencakup semua bagian sebelumnya dan dijelaskan dengan singkat.

Usulan kelengkapan :

->bagian 15.1

- sementara belum ada usulan untuk melengkapi bagian ini.

->bagian 15.2

- mungkin lebih baik bagian 15.2 diberi judul "Penjadwalan Asymmetric Multiprocessing" atau "Penjadwalan AMP" agar lebih sesuai dengan bagian selanjutnya.

->bagian 15.3

- mungkin di contoh OS modern yang mendukung SMP bisa ditambahkan lagi Windows Vista.

->bagian 15.4

- perlu dijelaskan sedikit mengenai apa itu workload karena istilah ini baru muncul pada bab ini. mungkin bisa ditambahkan informasi "workload (banyaknya task yang harus diselesaikan oleh prosesor)"

->bagian 15.5

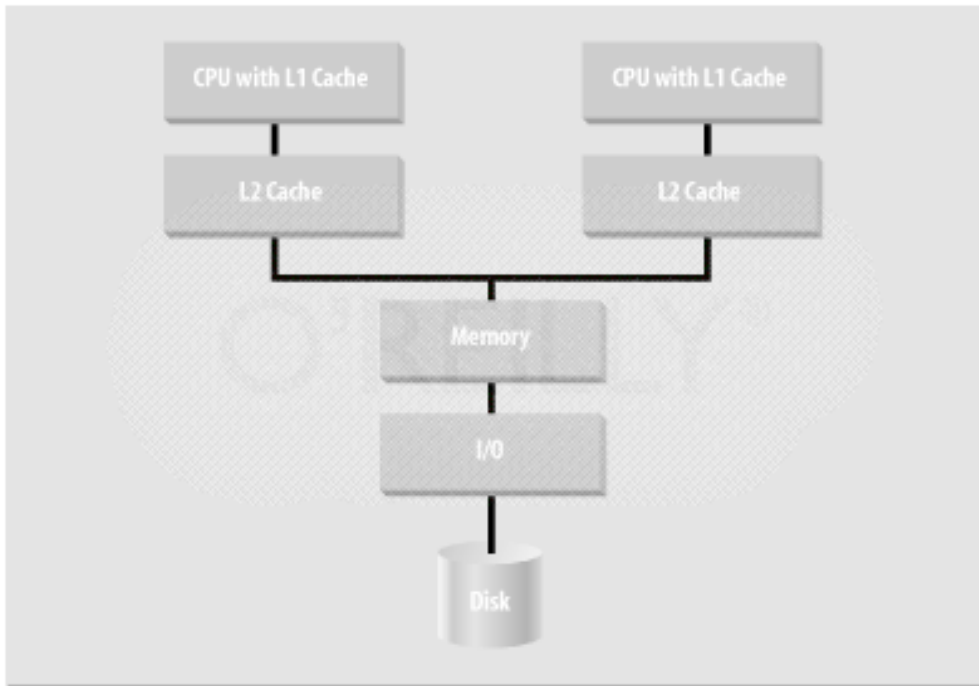
- kata chace seharusnya ditulis cache.
- penulisan symetric multithreading pada judul seharusnya ditulis symmetic multithreading.

->bagian 15.6

- seharusnya thread level paralelism ditulis thread-level parallelism.

->bagian 15.7

- mungkin perlu ditambahkan definisi multi processor yaitu penambahan prosesor yang diimplementasikan dalam suatu sistem untuk mempertinggi kinerja, kehandalan, kemampuan komputasi, paralelisme, dan keekonomisan.



Dibawah kami lampirkan hasil editan bab15 yaitu penjadwalan prosesor jamak.

Bab 15. Penjadwalan Prosesor Jamak

15.1. Pendahuluan

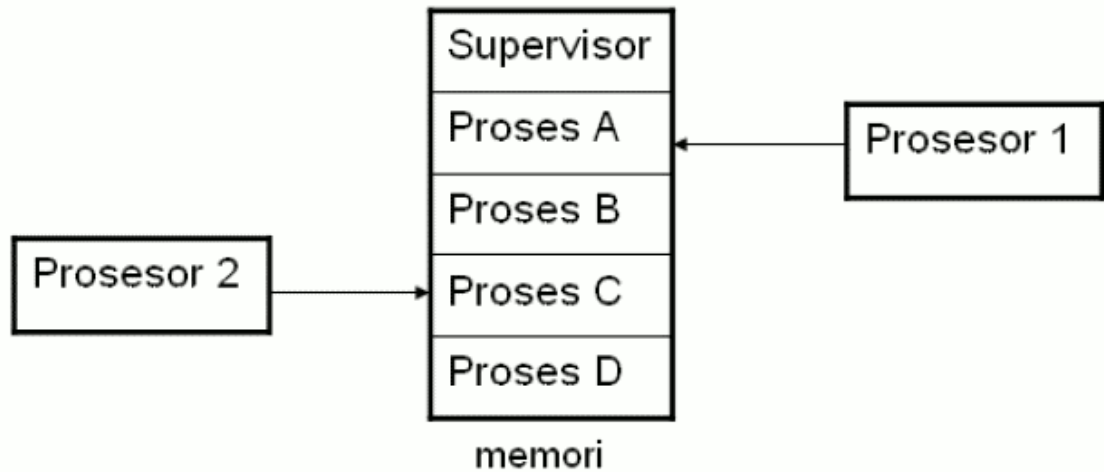
Untuk mempertinggi kinerja, kehandalan, kemampuan komputasi, paralelisme, dan keekonomisan dari suatu sistem, tambahan prosesor dapat diimplementasikan kedalam sistem tersebut. Sistem seperti ini disebut dengan sistem yang bekerja dengan banyak prosesor (prosesor jamak atau *multiprocessor*). Seperti halnya pada prosesor tunggal, prosesor jamak juga membutuhkan penjadwalan. Namun pada prosesor jamak, penjadwalannya jauh lebih kompleks dari pada prosesor tunggal karena pada prosesor jamak memungkinkan adanya *load sharing* antar prosesor yang menyebabkan penjadwalan menjadi lebih kompleks namun kemampuan sistem tersebut menjadi lebih baik. Oleh karena itu, kita perlu mempelajari penjadwalan pada prosesor jamak berhubung sistem dengan prosesor jamak akan semakin banyak digunakan karena kemampuannya yang lebih baik dari sistem dengan prosesor tunggal. Ada beberapa jenis dari sistem prosesor jamak, namun yang akan dibahas dalam bab ini adalah penjadwalan pada sistem prosesor jamak yang memiliki fungsi-fungsi prosesor yang identik (*homogenous*).

15.2. Penjadwalan *Asymmetric Multiprocessing* (Penjadwalan AMP)

Pendekatan pertama untuk penjadwalan prosesor jamak adalah penjadwalan *asymmetric multiprocessing* atau biasa disebut juga sebagai penjadwalan *master/slave*. Dimana pada metode ini hanya satu prosesor (*master*) yang menangani semua keputusan penjadwalan pemrosesan M/K, dan aktivitas sistem lainnya dan prosesor lainnya (*slave*) hanya mengeksekusi proses. Metode ini sederhana karena hanya satu prosesor yang mengakses struktur data sistem dan juga mengurangi *data sharing*.

Dalam teknik penjadwalan *master/slave*, satu prosesor menjaga status dari semua proses dalam sistem dan menjadwalkan kinerja untuk semua prosesor *slave*. Sebagai contoh, prosesor *master* memilih proses yang akan dieksekusi, kemudian mencari prosesor yang *available*, dan memberikan instruksi *start processor*. Prosesor *slave* memulai eksekusi pada lokasi memori yang dituju. Saat *slave* mengalami sebuah kondisi tertentu seperti meminta M/K, prosesor *slave* memberi interupsi kepada prosesor *master* dan berhenti untuk menunggu perintah selanjutnya. Perlu diketahui bahwa prosesor *slave* yang berbeda dapat ditujukan untuk suatu proses yang sama pada waktu yang berbeda.

Gambar 15.1. Multiprogramming dengan multiprocessor

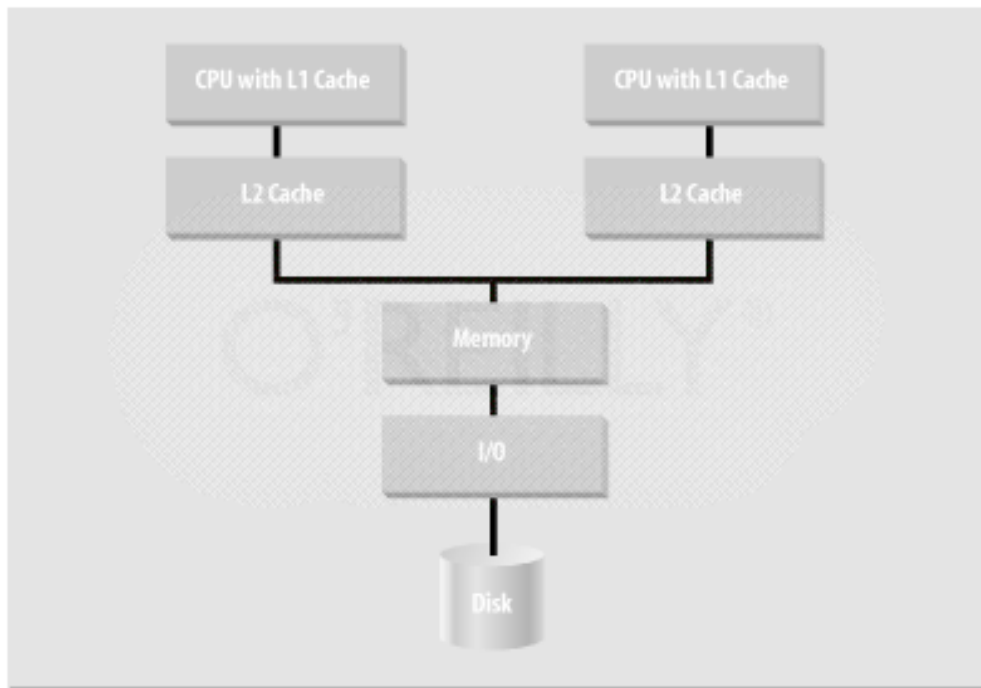


Gambar diatas mengilustrasikan perilaku dari *multiprocessor* yang digunakan untuk *multiprogramming*. Beberapa proses terpisah dialokasikan didalam memori. Ruang alamat proses terdiri dari halaman-halaman sehingga hanya sebagian saja dari proses tersebut yang berada dalam memori pada satu waktu. Hal ini memungkinkan banyak proses dapat aktif dalam sistem.

15.3. Penjadwalan *Symmetric Multiprocessing* (Penjadwalan SMP)

Penjadwalan SMP (*Symmetric multiprocessing*) adalah pendekatan kedua untuk penjadwalan prosesor jamak. Dimana setiap prosesor menjadwalkan dirinya sendiri (*self scheduling*). Semua proses mungkin berada pada antrian *ready* yang biasa, atau mungkin setiap prosesor memiliki antrian *ready* tersendiri. Bagaimanapun juga, penjadwalan terlaksana dengan menjadwalkan setiap prosesor untuk memeriksa antrian *ready* dan memilih suatu proses untuk dieksekusi. Jika suatu sistem prosesor jamak mencoba untuk mengakses dan meng-*update* suatu struktur data, penjadwal dari prosesor-prosesor tersebut harus diprogram dengan hati-hati; kita harus yakin bahwa dua prosesor tidak memilih proses yang sama dan proses tersebut tidak hilang dari antrian. Secara virtual, semua sistem operasi modern mendukung SMP, termasuk Windows XP, Windows 2000, Windows Vista, Solaris, Linux, dan Mac OS X.

Gambar 15.2. Multiprogramming dengan Symmetric Multiprocessing



Affinity dan Load Ballancing

Affinity

Data yang paling sering diakses oleh beberapa proses akan memadati *cache* pada prosesor, sehingga akses memori yang sukses biasanya terjadi di memori *cache*. Namun, jika suatu proses . berpindah dari satu prosesor ke prosesor lainnya akan mengakibatkan isi dari *cache* memori yang dituju menjadi tidak valid, sedangkan *cache* memori dari prosesor asal harus disusun kembali populasi datanya. Karena mahalnya *invalidating* dan *re-populating* dari cache, kebanyakan sistem SMP mencoba untuk mencegah migrasi proses antar prosesor sehingga menjaga proses tersebut untuk berjalan di prosesor yang sama. Hal ini disebut afinitas prosesor (*processor affinity*).

Ada dua jenis afinitas prosesor, yakni:

- *Soft affinity* yang memungkinkan proses berpindah dari satu prosesor ke prosesor yang lain, dan
- *Hard affinity* yang menjamin bahwa suatu proses akan berjalan pada prosesor yang sama dan tidak berpindah. Contoh sistem yang menyediakan *system calls* yang mendukung *hard affinity* adalah Linux.

15.4. Load Balancing

Dalam sistem SMP, sangat penting untuk menjaga keseimbangan *workload* (banyaknya task yang harus diselesaikan oleh prosesor) antara semua prosesor untuk memaksimalkan keuntungan memiliki *multiprocessor*. Jika tidak, mungkin satu atau lebih prosesor *idle* disaat prosesor lain harus bekerja keras dengan *workload* yang tinggi. *Load balancing* adalah usaha untuk menjaga *workload* terdistribusi sama rata untuk semua prosesor dalam sistem SMP. Perlu diperhatikan bahwa *load balancing* hanya perlu dilakukan pada sistem dimana setiap prosesor memiliki antrian tersendiri (*private queue*) untuk proses-proses yang berstatus *ready*. Pada sistem dengan antrian yang biasa (*common queue*), *load balancing* tidak diperlukan karena sekali prosesor menjadi *idle*, prosesor tersebut segera mengerjakan proses yang dapat dilaksanakan dari antrian biasa tersebut. Perlu juga diperhatikan bahwa pada sebagian besar sistem operasi kontemporer mendukung SMP, jadi setiap prosesor bisa memiliki *private queue*.

Ada dua jenis *load balancing*, yakni:

- *Push migration*, pada kondisi ini ada suatu *task* spesifik yang secara berkala memeriksa *load* dari tiap-tiap prosesor. Jika terdapat ketidakseimbangan, maka dilakukan perataan dengan memindahkan (*pushing*) proses dari yang kelebihan muatan ke prosesor yang *idle* atau yang memiliki muatan lebih sedikit.
- *Pull migration*, kondisi ini terjadi saat prosesor yang *idle* menarik (*pulling*) proses yang sedang menunggu dari prosesor yang sibuk.

Kedua pendekatan tersebut tidak harus *mutually exclusive* dan dalam kenyataannya sering diimplementasikan secara paralel pada sistem *load-balancing*.

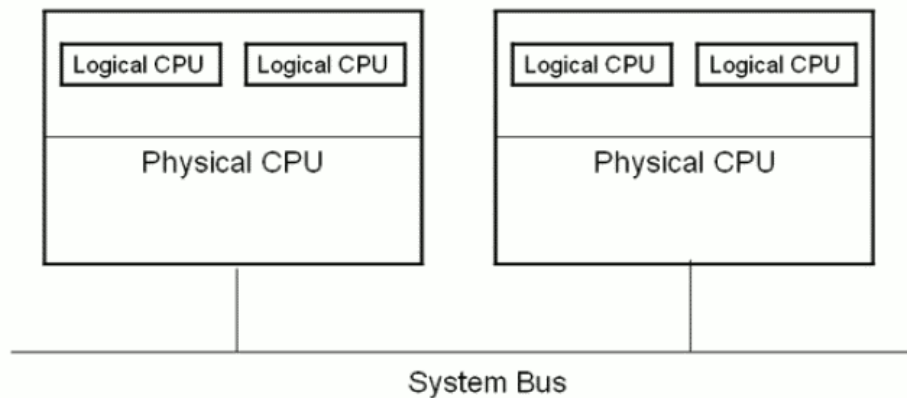
Keuntungan dari *affinity* berlawanan dengan keuntungan dari *load balancing*, yaitu keuntungan menjaga suatu proses berjalan pada satu prosesor yang sama dimana proses dapat memanfaatkan data yang sudah ada pada memori *cache* prosesor tersebut berkebalikan dengan keuntungan menarik atau memindahkan proses dari satu prosesor ke prosesor lain. Dalam kasus *system engineering*, tidak ada aturan tetap keuntungan yang mana yang lebih baik. Walaupun pada beberapa sistem, prosesor *idle* selalu menarik proses dari prosesor *non-idle* sedangkan pada sistem yang lain, proses dipindahkan hanya jika terjadi ketidakseimbangan yang besar antara prosesor.

15.5 Symmetric Multithreading

Sistem SMP mengizinkan beberapa *thread* untuk berjalan secara bersamaan dengan menyediakan banyak *physical processor*. Ada sebuah strategi alternatif yang lebih cenderung untuk menyediakan *logical processor* daripada *physical processor*. Strategi ini dikenal sebagai SMT (*Symmetric Multithreading*). SMT juga biasa disebut teknologi *hyperthreading* dalam prosesor intel.

Ide dari SMT adalah untuk menciptakan banyak *logical processor* dalam suatu *physical processor* yang sama dan mempresentasikan beberapa prosesor kepada sistem operasi. Setiap *logical processor* mempunyai *state* arsitekturnya sendiri yang mencakup *general purpose* dan *machine state register*. Lebih jauh lagi, setiap *logical processor* bertanggung jawab pada penanganan interupsinya sendiri, yang berarti bahwa interupsi cenderung dikirimkan ke *logical processor* dan ditangani oleh *logical processor* bukan *physical processor*. Dengan kata lain, setiap *logical processor* men- *share resource* dari *physical processor*-nya, seperti *cache* dan *bus*.

Gambar 15.3. Symetric Multithreading



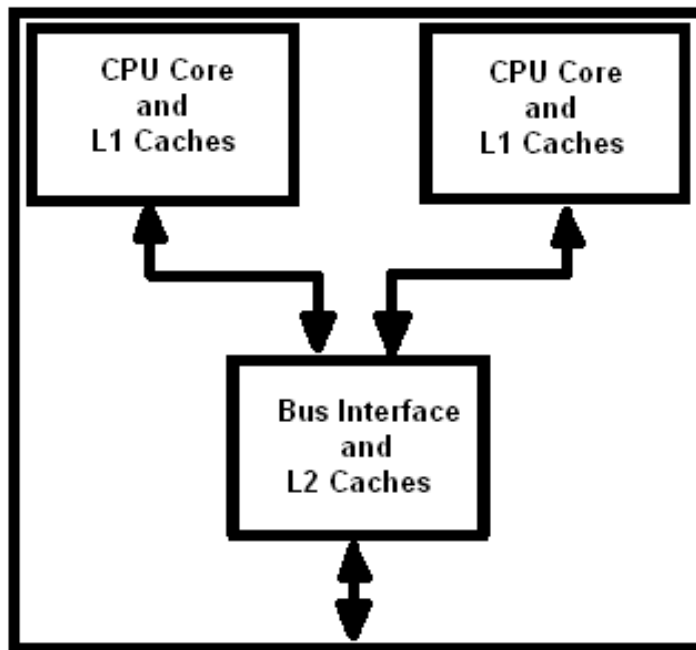
Gambar di atas mengilustrasikan suatu tipe arsitektur SMT dengan dua *physical processor* dengan masing-masing punya dua *logical processor*. Dari sudut pandang sistem operasi, pada sistem ini terdapat empat prosesor.

Perlu diketahui bahwa SMT adalah fitur yang disediakan dalam *hardware*, bukan *software*, sehingga *hardware* harus menyediakan representasi *state* arsitektur dari setiap *logical processor* sebagaimana representasi dari penanganan interupsinya. Sistem operasi tidak perlu didesain khusus jika berjalan pada sistem SMT, akan tetapi performa yang diharapkan tidak selalu terjadi pada sistem operasi yang berjalan pada SMT. Misalnya, suatu sistem memiliki 2 *physical processor*, keduanya *idle*, penjadwal pertama kali akan lebih memilih untuk membagi *thread* ke *physical processor* daripada membaginya ke *logical processor* dalam *physical processor* yang sama, sehingga *logical processor* pada satu *physical processor* bisa menjadi sibuk sedangkan *physical processor* yang lain menjadi *idle*.

Multicore

Multicore microprocessor adalah kombinasi dua atau lebih prosesor independen ke dalam sebuah *integrated circuit* (IC). Pada umumnya, *multicore* mengizinkan perangkat komputasi untuk memeragakan suatu bentuk *thread-level paralelism* (TLP) tanpa mengikutsertakan banyak prosesor terpisah. TLP lebih dikenal sebagai *chip-level multiprocessing*.

Gambar 15.4. *Chip CPU dual-core*



Keuntungan:

- Meningkatkan performa dari operasi *cache snoop (bus snooping)*. *Bus snooping* adalah suatu teknik yang digunakan dalam sistem pembagian memori terdistribusi dan *multiprocessor* yang ditujukan untuk mendapatkan koherensi pada *cache*. Hal ini dikarenakan sinyal antara CPU yang berbeda mengalir pada jarak yang lebih dekat, sehingga kekuatan sinyal hanya berkurang sedikit. Sinyal dengan kualitas baik ini memungkinkan lebih banyak data yang dikirimkan dalam satu periode waktu dan tidak perlu sering di-repeat .
- Secara fisik, desain CPU *multicore* menggunakan ruang yang lebih kecil pada PCB (*Printed Circuit Board*) dibandingkan dengan desain *multichip* SMP.
- Prosesor *dual-core* menggunakan sumber daya lebih kecil dibandingkan dengan sepasang prosesor *dual-core*.
- Desain *multicore* memiliki resiko *design error* yang lebih rendah daripada desain *single-core*.

Kerugian:

- Dalam hal sistem operasi, dibutuhkan penyesuaian pada *software* yang ada untuk memaksimalkan kegunaan dari sumber daya komputasi yang disediakan oleh prosesor *multicore*. Kemampuan prosesor *multicore* untuk meningkatkan performa aplikasi juga bergantung pada jumlah penggunaan *thread* dalam aplikasi tersebut.
- Dari sudut pandang arsitektur, pemanfaatan daerah permukaan silikon dari desain *single-core* lebih baik daripada desain *multicore*.
- Pengembangan *chip multicore* membuat produksinya menjadi turun karena bertambahnya tingkat kesulitan untuk mengatur suhu pada *chip* yang padat.

Pengaruh *multicore* terhadap *software*

Keuntungan *software* dari arsitektur *multicore* adalah kode-kode dapat dieksekusi secara paralel. Dalam sistem operasi, kode-kode tersebut dieksekusi dalam *thread-thread* atau proses-proses yang terpisah. Setiap aplikasi pada sistem berjalan pada prosesnya sendiri sehingga aplikasi paralel akan mendapatkan keuntungan dari arsitektur *multicore*. Setiap aplikasi harus tertulis secara spesifik untuk memaksimalkan penggunaan dari banyak *thread*.

Banyak aplikasi *software* tidak dituliskan dengan menggunakan *thread-thread* yang *concurrent* karena tingkat kesulitan yang tinggi dalam pembuatannya. *Concurrency* memegang peranan utama dalam aplikasi paralel yang sebenarnya.

Langkah-langkah dalam mendesain aplikasi paralel adalah sebagai berikut:

1. ***Partitioning***. Tahap desain ini dimaksudkan untuk membuka peluang awal pengekseskuan secara paralel. Fokus dari tahap ini adalah mempartisi sejumlah besar tugas dalam ukuran kecil dengan tujuan menguraikan suatu masalah menjadi butiran-butiran kecil.
2. ***Communication***. Tugas-tugas yang telah terpartisi diharapkan dapat langsung dieksekusi secara parallel. Akan tetapi, pada umumnya tidak bisa, karena eksekusi berjalan secara independen. Pelaksanaan komputasi dalam satu tugas membutuhkan asosiasi data antara masing-masing tugas. Kemudian data harus berpindah-pindah antar tugas dalam melangsungkan komputasi. Aliran informasi inilah yang dispesifikasi dalam fase *communication*.

3. **Agglomeration.** Pada tahap ini kita pindah dari sesuatu yang abstrak ke sesuatu yang konkret. Kita tinjau kembali kedua tahap di atas dengan tujuan untuk mendapatkan algoritma pengeksekusian yang lebih efisien. Kita pertimbangkan juga apakah perlu untuk menggumpalkan (*agglomerate*) tugas-tugas pada fase *partition* menjadi lebih sedikit, dengan masing-masing tugas berukuran lebih besar.
4. **Mapping.** Dalam tahap yang keempat dan terakhir ini, kita menspesifikasi di mana tiap tugas akan dieksekusi. Masalah *mapping* ini tidak muncul pada *uniprocessor* yang menyediakan penjadwalan tugas.

Pada sisi *server*, prosesor *multicore* menjadi ideal karena *server* mengizinkan banyak *user* untuk melakukan koneksi ke *server* secara simultan. Oleh karena itu, *Web server* dan *application server* mempunyai *throughput* yang lebih baik.

15.7. Rangkuman

Penjadwalan *asymmetric multiprocessing* atau penjadwalan *master/slave* menangani semua keputusan penjadwalan, pemrosesan M/K, dan aktivitas sistem lainnya hanya dengan satu prosesor(*master*). Dan prosesor lainnya (*slave*) hanya mengeksekusi proses.

SMP (*Symmetric multiprocessing*) adalah pendekatan kedua untuk penjadwalan prosesor jamak. Dimana setiap prosesor menjadwalkan dirinya sendiri(*self scheduling*).

Load balancing adalah usaha untuk menjaga *workload* terdistribusi sama rata untuk semua prosesor dalam sistem SMP. *Load balancing* hanya perlu untuk dilakukan pada sistem dimana setiap prosesor memiliki antrian tersendiri (*private queue*) untuk proses-proses yang akan dipilih untuk dieksekusi. Ada dua jenis *load balancing*: *push migration* dan *pull migration*. Pada *push migration*, ada suatu *task* spesifik yang secara berkala memeriksa *load* dari tiap-tiap prosesor, jika terdapat ketidakseimbangan, maka dilakukan perataan dengan memindahkan (*pushing*) proses dari yang kelebihan muatan ke prosesor yang *idle* atau yang memiliki muatan lebih sedikit. *Pull migration* terjadi saat prosesor yang *idle* menarik (*pulling*) proses yang sedang menunggu dari prosesor yang sibuk. Kedua pendekatan tersebut tidak harus *mutually exclusive* dan dalam kenyataannya sering diimplementasikan secara paralel pada sistem *load-balancing*.

Afinitas prosesor (*processor affinity*) adalah pencegahan migrasi proses antar prosesor sehingga menjaga proses tersebut tetap berjalan di prosesor yang sama. Ada dua jenis afinitas prosesor, yakni *soft affinity* dan *hard affinity*. Pada situasi *soft affinity* ada kemungkinan proses berpindah dari satu prosesor ke prosesor yang lain. Sedangkan *hard affinity* menjamin bahwa suatu proses akan berjalan pada prosesor yang sama dan tidak berpindah. Contoh sistem yang menyediakan *system calls* yang mendukung *hard affinity* adalah Linux.

SMT (*Symmetric Multithreading*) adalah strategi alternatif untuk menjalankan beberapa *thread* secara bersamaan. SMT juga biasa disebut teknologi *hyperthreading* dalam prosesor intel.

Multicore microprocessor adalah kombinasi dua atau lebih *independent processor* kedalam sebuah *integrated circuit (IC)*. *Multicore microprocessors* mengizinkan perangkat komputasi untuk memeragakan suatu bentuk *thread level parallelism (TLP)* tanpa mengikutsertakan banyak *microprocessor* pada paket terpisah. TLP lebih dikenal sebagai *chip-level multiprocessing*.

Rujukan

[Silberschatz2002] Abraham Silberschatz, Peter Galvin, dan Greg Gagne. 2002. *Applied Operating Systems*. Sixth Edition. John Wiley & Sons.

[Silberschatz2005] Avi Silberschatz, Peter Galvin, dan Greg Gagne. 2005. *Operating Systems Concepts*. Seventh Edition. John Wiley & Sons.

[WEBWiki2007] Wikipedia. 2007. *Multicore(Computing)* – <http://en.wikipedia.org/wiki/Multicore> . Diakses 27 Februari 2007.

http://www.oreilly.com/catalog/9780596514549/figs/I_mediaobject12_d1e21060.png . Diakses 4 Maret 2008.

http://searchdatacenter.techtarget.com/sDefinition/0,,sid80_gci970333,00.html . Diakses 4 Maret 2008.

Hak Cipta © Mahasiswa Fakultas Ilmu Komputer Universitas Indonesia, peserta mata kuliah Sistem Operasi semester genap 2008

Andri Satria (0606104196)

Freddy Setiawan (0606104252)

Hendy Kusumo Nugroho (0606101420)

R. Brahmastro K (1205000746)

Silakan menyalin, mengedarkan, dan/atau, memodifikasi dokumen ini.