Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors

by A. Tucker and A. Gupta

- Idea: application controls the number of runnable processes dynamically
- Implementation on a shared memory machine
- If processes $>$ processors then performance degradation
  - preemption while in critical section
  - producer/consumer
  - context switching
  - cache corruption
- Increase/reduce the number of processes per application as a function of system load
- Scheduler controlled

- Application issues
  - when is it safe to preempt?
  - implementation on top of a thread library package
  - transparent to the programmer
- System issues
  - controlled approach
  - central server
- Algorithm
  1. assumption: root process for each application
  2. root process send its ID to the controller
  3. starts executing and periodically polls server
  4. controls its dependent processes by waking/suspending
- Results/Advantages/Disadvantages

Scheduling Techniques for Concurrent Systems

by John K. Ousterhout

- Introduces the notion of *coscheduling*
- Deals with interprocess communication
- Introduces *process working set*
- How to avoid *process thrashing*
  - determine process working set
  - coschedule those working sets
- Three algorithms
  - matrix method
  - continuous algorithm
  - undivided algorithm

Assumptions/Definitions/Goals

- No process migration
- Task force: processes from the same job (gang)
- Task force fragmentation
- Maximize the average number of processors executing coscheduling processes
- P processors that can multiplex across at most Q processes

The Matrix Method

- Matrix of Q rows and P processors
- Allocation: scan each row, until you find a single row where the task force fits
- Scheduling: round-robin across rows
- What if one or more process slots in a row are empty or contain blocked processes?
  - it scans the column for another *runnable* process and runs its as an *alternate*
- Problems...
  - fragmentation: unused slots per row, poor binpacking
  - what if a task force does not fit on a row?
  - alternate selection: beats the purpose of coscheduling

The Continuous Algorithm

- Denser packing
- Unfolds the matrix $\rightarrow$ single sequence of scheduling slots
- Sliding window of size P
- Allocation: Slide the window, till a window position is found that contains the entire task force $\rightarrow$ reduces fragmentation
- Scheduling: move the window as many slots so that the leftmost slot is always the leftmost slot of a task force
  - every task force is scheduled at least once in every sweep
  - every window coschedules at least one task force
- Drawbacks
  - not fair: small *vs* large task forces
  - fragmentation at high loads

The Undivided Algorithm

- Does not allow a task force to be divided between holes
- Less fragmentation
- Better under heavy load

Analysis Using Simulation

- Parameters
- Measures
  - response time
  - $\frac{processes\ \ executing\ \ coscheduled}{processors\ \ with\ \ runnable\ \ processes}$

Summary

- Costs
- Density of packing
- Fragmentation
- Best: undivided

Revisit

- Performance in an environment with diverse demands
- Problems with batch processing and space partitioning
- Process Control
  - must be written as multithreaded applications
  - operating point effect
  - critical number of processors
- Independent time sharing versus coordinated
- Coscheduling
  - processes ¿ processors
  - alternate selection
  - overheads

- Alternates
  - distributed hierarchical control
  - dynamic and predictive coscheduling
  - patterns of message exchanges
  - history of communication “correspondents”
- Hardest part: performance comparison