Scheduling Algorithm Notes July 14, 1969

A possible simplified algorithm

1. All scheduling times are CPU times; (accounting and ready message times are done separately)
2. Running eligible, and ready processes are kept in a single priority
3. Whenever a CPU timer is set, the value is to t_{emin}, a constant.
4. Preemption only occurs at timer runouts or at calls to block.
5. Wait and Notify do not modify the priority list nor reassign elligibility; rather these mechanism are responsible for multiprogramming.

Definitions

t_i = total time used since interaction began in previous scheduling levels.

t_s = time at this scheduling level

t_{emin}, t_{emax} = minimum time a process runs once started (constant nominally 4 sec.)

t_{smax} = maximum time spend being saved at the head of lowest priority queue before being placed at the end of the queue, (constant, nominally 16 sec.)

t_{used} = time a process has used since the timer was last set.

July 14, 1969

Wakeup(k)

add k^{th} process to priority list according to t_i^k

return;

Block

if isw then $t_i, t_s = 0$;

else do;

$t_s = t_s + t_{used(timer)}$;

if $t_s > t_i$ then do;

if $t_s \geq t_{s_{max}}$ then $t_i = t_{s_{max}} - t_{e_{min}}$

$t_i = t_i + t_s$;

$t_s = 0$; end; end;

Remove own elig and ~~reserve from~~ priority list

Confer elig ~~first~~ highest priority process that can make elig.

Network

timer = $t_{e_{min}}$;

return;

Timer Runout

$t_s = t_s + t_{used(timer)}$;

if $t_s > t_i$ then do;

if $t_s \geq t_{s_{max}}$ then $t_i = t_{s_{max}} - t_{e_{min}}$;

$t_i = t_i + t_s$;

$t_s = 0$;

put self in priority list by t_i; end

~~Remove~~ own elig.

Confer elig. on highest priority process that can make elig.

getwork

timer = $t_{e_{min}}$;

return;