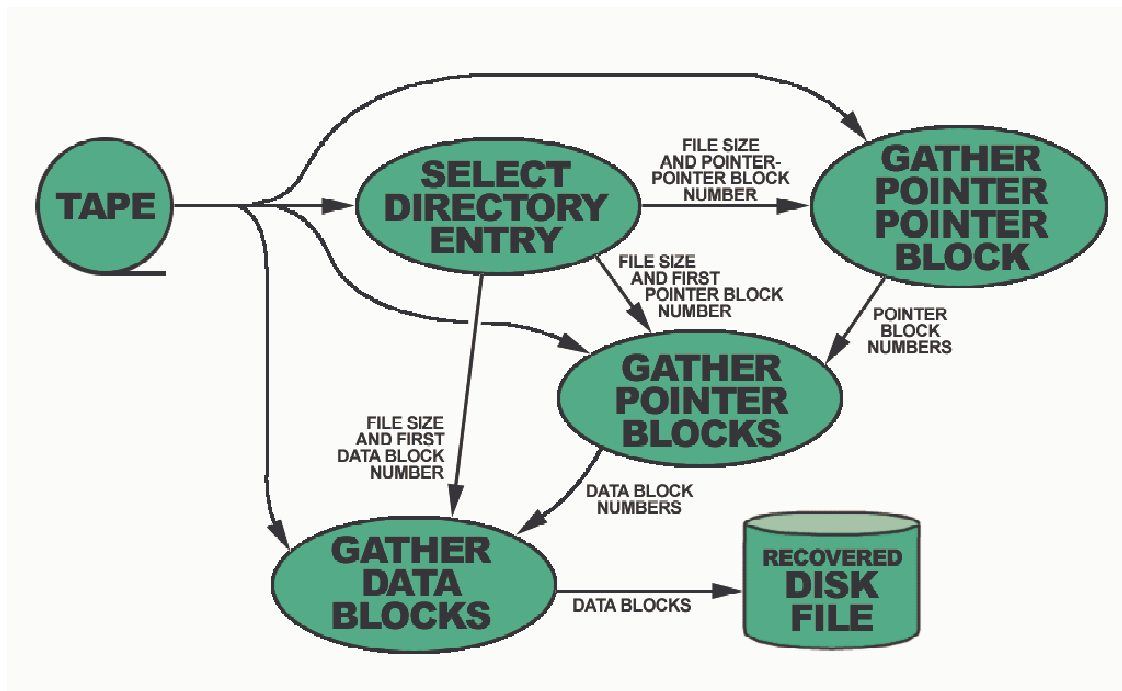


E-GENTING PROGRAMMING COMPETITION, 2004

WORKSHOP HANDOUT, WEEK 4, VERSION 1

1 BACKGROUND

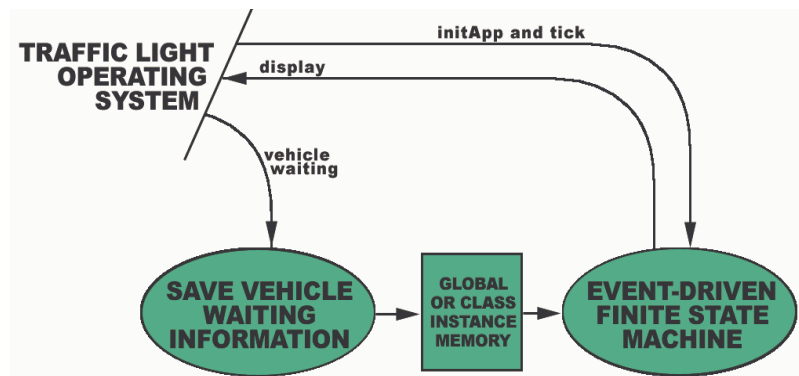
1.1 Dataflow Revisited



1.2 Process Implementations

- a program;
- a thread;
- a class;
- a function;
- an interrupt service routine or a group of interrupt service routines;
- in-line code;
- shared memory;
- a pipe.

1.3 Pseudo-Code



Event-driven finite state machine:

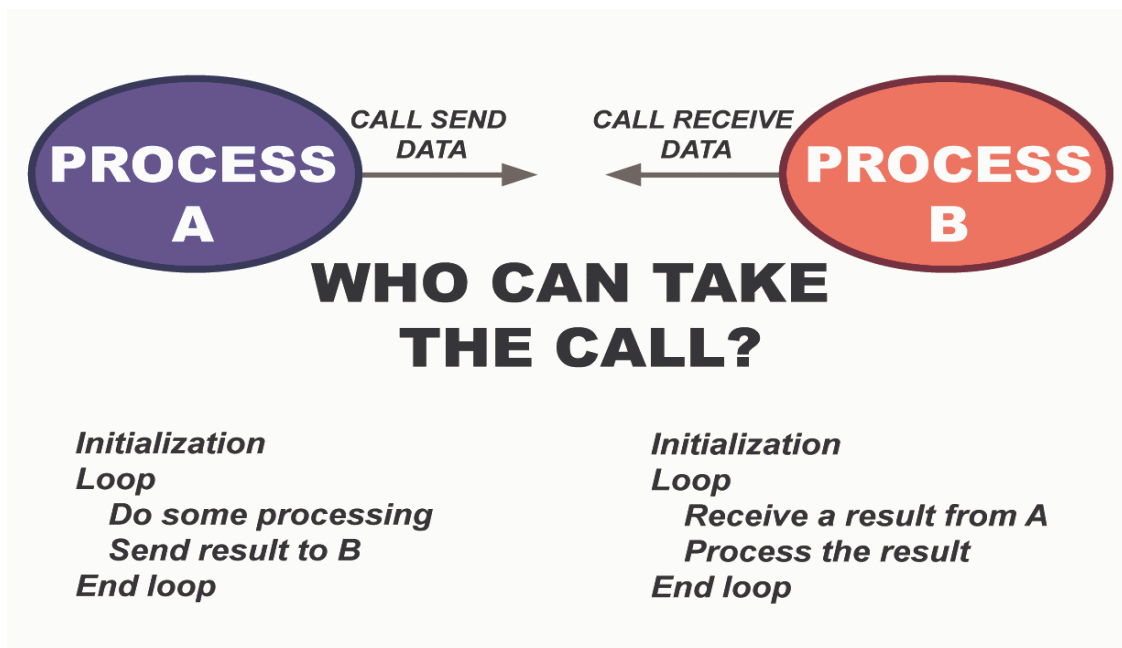
1. Wait for initiation ['UNINITIATED' state];
2. branch = NORTH;
3. Loop:
 - a. Display green light combination (branch);
 - b. For green time (branch):
 - i. Wait for a clock tick ['GREEN' state];
 - c. Display amber light combination (branch);
 - d. For amber time (branch):
 - i. Wait for a clock tick ['AMBER' state];
 - e. Read vehicle waiting information;
 - f. branch = Next (branch, vehicle waiting information);
4. End loop.

1.4 Event-Driven Finite State Machine

Event	State	Procedure
Initiate	UNINITIATED	<ol style="list-style-type: none"> 1. branch = NORTH; 2. Display green light combination (branch); 3. state = GREEN.
Clock tick	GREEN	<ol style="list-style-type: none"> 1. If green time(branch) has elapsed: <ol style="list-style-type: none"> a. Display amber light combination (branch); b. state = AMBER; 2. End if.

Clock tick	AMBER	<ol style="list-style-type: none"> 1. If amber time(branch) has elapsed: <ol style="list-style-type: none"> a. Read vehicle waiting information; b. branch = Next (branch, vehicle waiting information); c. Display green light combination (branch); d. state = GREEN; 2. End if.
------------	-------	---

1.5 The Client-Client Standoff



Convert Process B into server mode. Process A can then call the 'Result' method in Process B to send a result to Process B.

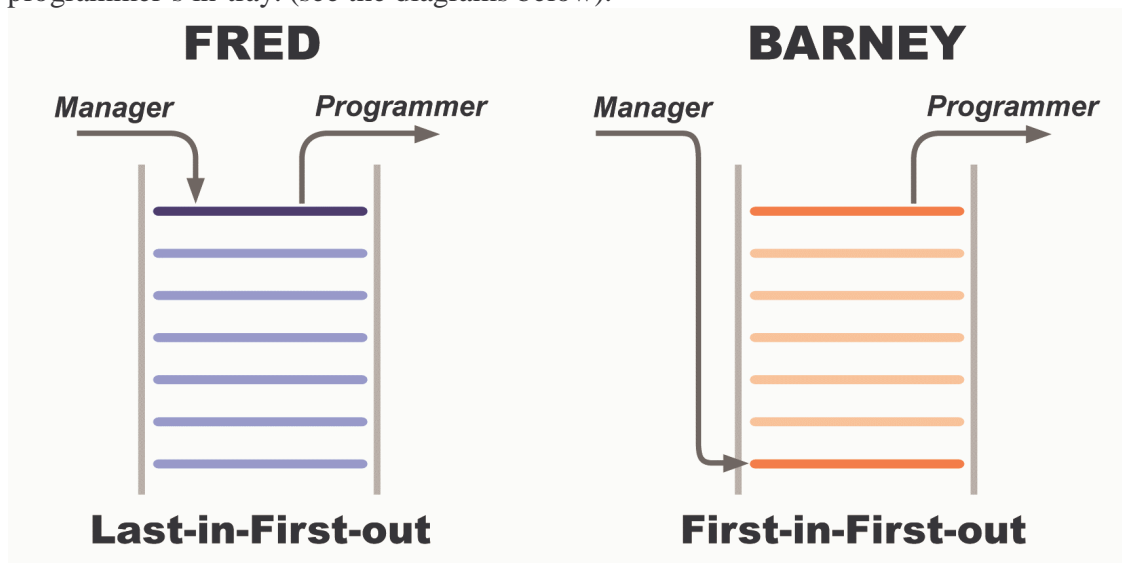
Stimulus	State	Processing
Initiate	UNINITIATED	<ol style="list-style-type: none"> 1. Initialisation. 2. state = READY.
Result	READY	<ol style="list-style-type: none"> 1. Process the result.

However, it is worthwhile considering a little optimisation.

<ol style="list-style-type: none"> 1. Initialisation of Process A. 2. Initialisation of Process B. 3. Loop <ol style="list-style-type: none"> a. Do the processing from Process A. b. Process the result as per Process B. 4. End loop.
--

2 EXERCISE

Consider a work situation that involves two managers called Fred and Barney and two programmers. The first programmer receives instructions from Fred whereas the second programmer receives instructions from Barney. These two teams work on separate projects. There are two separate 'in-trays' from which the two programmers receive their respective instructions. When each programmer finishes a task, he takes the next job from the top of his in-tray. Fred's practice is to place the next work order on top of his programmer's in-tray whereas Barney always places the next task at the bottom of his programmer's in-tray. (see the diagrams below).



Write a computer simulation program either in C, C++ or Java to obtain an estimate of the average and standard deviation of the total delivery time per task for each of the processing sequences described above. The total delivery time refers to the sum of the waiting time and the programming time. The 'in tray' must be represented as a linked list for both cases.

For the purpose of this exercise, assume that the inter-task arrival time is evenly distributed between 4 and 7 days in real time for both cases. Similarly, assume that the time required by each programmer to accomplish each programming task is evenly distributed between 3.9 and 6.9 days in real time. Obtain the required estimates based on 200,000 simulated tasks.

For those who did not study Statistics, the standard deviation of the total delivery time is given by the following expression:

$$S = \sqrt{\frac{\sum_{i=1}^n x_i^2}{n} - \bar{X}^2}$$

where x_i , \bar{X} and n denote respectively the total delivery time for the i th task, average total delivery time per task and number of tasks.