CONCURRENT PROGRAMMING OF INTELLIGENT ROBOTS

Yutaka Kanayama

University of Tsukuba
Sakura, Ibaraki 305 Japan

ABSTRACT

Real time intelligent robots usually consist of more than one processing unit (pu) to ensure parallel operation of several functions. Each pu in a robot executes repetitive monitoring and controlling operations as well as information exchange to and from other pu's. Since timing of each operation is independent of others,if the robot operating software supports concurrent process facilities, it would be helpful in robot programming.

A self-contained robot "Yamabico 9" has been constructed to be a tool for investigating how a mobile robot understands the outer world. In order to support software production on the robot, Robot Control System (RCS) has been implemented, including simple job commands and a supervisor call (SVC) system. The concurrent process monitor is a part of RCS and some of SVC's are for these facilities. The monitor adopts a "message sending" method to synchronize execution of two processes and to exchange information between processes and pu's. An example of a concurrent process program, "walk along left walls", is given to demonstrate the describing power of our system.

Introduction

A human can walk, speak, think, look around, hear and handle things at the same time. Like human beings, multifunctional intelligent robots consist of several independent processing units (pu) that work in parallel.

Even for a robot with multiple pu's, the operation of each pu may be quite complex. An example of concurrent operations in a pu of a robot is shown in Figure 1. Processes 2 and 5 are semi-autonomous ones which monitor outer environments and/or control the effectors. Process 1 and 6 are, respectively, input and output processes of this pu. Process 2 conditionally wakes up process 3. Process 4 is a subprocess of process 2. The total of each process's occupation rate should be less than 100 % of the time of the processor.

The concept of concurrent process in multiprocessor systems is described in Figure 2. Concurrent process facilities are especially useful in multiprocessor systems,because communication between pu's has to be supported.

It is very difficult to program such a robot in conventional sequential programming languages. Hierarchical subroutining does not help us for the purpose, because each subroutine occupies 100% of processor time for a period and what we want is interleaved execution of several tasks. Several mechanisms have been proposed for process synchronization and mutual exclusion of shared resources. Semaphore [1], monitor [2], csp [3] and rendezvous *[4]* are examples. We propose a simple mechanism, sending messages, because' robot programming has the following characteristics:

(1) A robot system is not a multiuser one. Therefore task switching at a definite time slice is not necessary and is even harmful.

(2) Processes need not be dynamically generated nor killed. Each process is considered to be a portion of the robot's intelligence and just exists on the robot. Some processes are usually in "sleep" state and waked up when needed.

(3) The necessity for sharing memory or other resources is not high in robot programming. Empirical facts tell us that a pu in a robot should be independent of every other as far as possible and should send a small amount of condensed information to others. A big amount of data, such as map data or scene data, can be supervised by only one process PO. If another process P wants to use that, it just asks a question to PO which returns the answer to P. Human information processing subsystems do not seem to share any common memory, and each one serially exchanges a small number of signals with each of the others.

II Hardware and Software of Yamabico 9

We had already reported on the Yamabico family of selfcontained mobile robots [5][6][7], and the most recent member is Yamabico 9 which has been constructed to be a tool for investigating how an autonomous machine can understand its two-dimensional environment (See Figure 3). This is an enhanced version of Yamabico 3.1 in the sense that the cpu of the brain has been changed from 6802 to 6809 and a 6809 processor system has been added to the supersonic range finding system

to be an independent processing unit. [8]

Our ultimate objective with Yamabico 9 is to implement a world-understanding intelligence with a map data base. We found that to implement a well organized operating system on the robot is inevitable for our future work, because if the efficiency of software production is low, it is almost impossible to complete software developing projects. The operating system is called Robot Control System, RCS. The pricipal functions of RCS are as follows [9"].

(I) To support software production in 6809 assembler language.

(?) To support simple Job control commands.

(3) To support a rich set of supervisor calls, SVC.

(4) To support concurrent process control using SVCs..

The size of RCS is about 8K bytes. Programs and data in RCS have the structure of a "memory module" in 0S-9 software system [I0]["III. Each program produced by OS-9 is relocatable and reentrant, so that the same text may be used by more than one process. RCS and some basic user's programs are implemented on the ROM in the self-contained robot. When a user constructs a program, it is loaded from OS-9 on the base computer-system to the robot via a communication channel.

Supervisor calls in RCS are in fact a combination of a software instruction (SWI) *of* 6809, a one parameter byte in instruction stream Just after the SWI instruction, and in some cases one or more input parameters that have been set on registers.

1II Concurrent Process Facilities

We adopted the "send message" function for process synchronization and mutual exclusion in common resource access. Several message sending schemes embedded in high level languages have been proposed [3][I2][I3], but we propose another one in which the protocol of RCS is very simple and buffers are dynamically controlled by RCS itself.

RCS and user programs are composed of processes. Each process has a distinct process identifier pid, whose length is one byte. Each process takes one of the following three states (See Figure 4):

- active
  - running : the cpu is executing this process.
  - ready : this process is ready to run, but there is a running one which has a higher priority.
- waiting : this process is waiting for a massage from another one.

Each process has a unique working area which contains a process descriptor which is under supervision of the concurrent process monitor, a part of RCS.

A process can go into the wait state by using "wait" SVC to free the processor, but can't "wake up"by itself from wait state. If there exists more than one active process, the highest priority one is selected to be in running state by the dispatcher routine in RCS.

A process can send a message in a buffer to another process by using a "send" SVC. The length of a message should be from 0 to 255. Each message buffer is allocated by RCS if requested, and it is freed after the message is processed by the receiver.

The functions stated above are controlled by using several SVC's, and their parameters are set on registers A, B, X and Y in the microprocessor MC6809 • For the details see [9].

IV Concurrent Programmi ng

When we write robot software in the RCS environment, all segments of procedures are in the form of processes. When the robot is turned on, after the process descriptors are initialized all processes go into the active state, and almost all of them in turn into the wait state.

Tf a process has to be "called", we use a "wake up" Job command, instead of using an "execute" command. RCS has no "execute" commands, because in a sense all processes are executed all the time. In sequential programming, subroutines are distributed in time, but in concurrent programming processes are distributed in space.

In our programming, a task is divided into one main process and several subprocesses. For example, in Figure 5, P is a main process and, P1, P-11 and P2 are subprocesses. When P is to direct P11 to perform a subtask, P Just send a message M to P11. That message M is called an input message for P1_ and an output message for P. P is called a parent of P1_ and P1 a child of P. A parent always knows its chiliren's process id (pid), but a child does not know or need not know its parent's name beforehand. When a child is called, the parent's pid is given on register A, which can be kept for later use.

Input messages are used to hand the parent's intention to the child. Some of them are "start", "stop" and "change parameter" messages. The first byte of each message is called the key of the message, which classifies its purpose.

V Example

To demonstrate how the concurrent process

facilities work, we will show a simple user's process called "Walk Along Left Walls", abbreviated WALW. This process makes Yamabico 9 walk in a building as though it were following a wall with its "left hand", and is useful for going out of a maze (See Figure *(>)*.

This process is more sophisticated than it looks. We divide it into one main process WALW, two child processes FLW and MNTF and one grandchild process CD. The whole structure is the same as Figure 5. Here we will describe the four processes.

A. Walk Along Left Walls (WALW) process

This is the main process.

(I) Input messages:

< 0 >	start	from RCS etc
< 1 >	stop	from RCS etc
< 1 wwl0 wwll ss	section	from FLW

(2) Output messages:

< 0 >	start	to FLW
< 0 >	start	to MNTF
< 1 >	stop	to FLW
< l >	stop	to MNTF
< 2 wl >	change distance	to FLW

(3) Functions:

a) At first, the robot looks around to find the nearest wall. If there are no walls, it stops.
b) Makes a turn so that it looks at the wall on the left side.
c) Begins to walk straight.
d) Wakes up FLW and MNTF.
e) The following tasks el ^ e5 are selectively repeated.
el) If this process receives a stop message, quits walking and go to wait state.
e2) If this process accepts an abnormal message from FLW or MNTF, quits walking and go to wait state.
e3) If the left wall disappears, continue to go another 100cm, turn left and again walk straight.
ek) If there is a wall to the front and there is no wall to the right, turn right and walk straight again.
e5) If there are walls to the front and to the right, make the robot stop, turn 180 and begin to walk straight.

The marks e3, *e4* and e5 in Figure 6 correspond to the tasks named above.

B Follow Left Walls (FLW) process

This process can be called by any process and works under any parent, and is one of the "intrinsic" or "basic" processes of RCS. This has a child process CD. When FLW is called by WALW, it cooperates with the parent and monitors the left and front walls continuously. The robot tries to be on an imaginary reference line. (See Figure T)

(I) Input messages:

< 0 >	start
< 1 >	stop
< 2 t >	designate cycle time
< >	finished from CD

(2) Output messages

<0 wwl0 wwll ss>	tell the parent about the change of walls
<0 d>	to CD

(3) Function:

a) At first if there is a flat wall, go straight and define the reference line (initialize WWL00).
b) The following tasks are selectively repeated.
bl) If the distance to the left wall suddenly changes, send a message to the parent and update WWL00.
b2) If the left wall disappears, send a message to the parent.
b3) If a "change parameter" message comes in, change the parameter.
b4) Otherwise, if the expected position ahead is- significantly away from the reference line, call CD to change the direction.

C. Monitor Front (MF) process

This is a simple process to monitor any obstacle in front, and send a warning to the parent.

(1) Input messages:

< 0 >	start
< 1 >	stop
< 2 ssf >	designate minimum distance (the default value=30cm)

(2) Output messages

< 0 ssf >	stationary obstacle found
< 1 ssf >	moving obstacle found

(3) Function:

Monitor the distance to the front all the time and determine if the object is stationary or not; if the distance is less than the given value, send a message to the parent. That discriminating facility is a part of the sonic range finding system of Yamabico *[14]*.

D. Change Direction (CD) process

This process is called when Yamabico 9 is walking and changes its direction by a given value. The function of the "turn" command of the Yamabico locomotion System is fully utilized [6].

(1) Input messages:

<0 d>	start to change the direction by d. The value may be negative.
< 1 >	stop

(2) Output messages:

< >	finished, to the parent (null message)

(3) Function:

Terminate the currently executing command in the locomotion system and send two motion commands so that the heading direction is changed by d. Ask the leg to send a message as soon as the locus becomes straight again; the process then sends a "finished" message to the parent. This process is running when Yamabico is on a curve between A and C (See Figure 8).

ACKNOWLEDGEMENT

The author thanks Professors S. Yuta of University of Tsukuba and J. Iijima of the University of Electro-Communications for their research and development effort in the basic Yamabico 9 system.

REFERENCES

[I] Dijkstra, E. W., The Structure of the T.H.E. Multiprogramming System, CACM, vol.11, no.5, pp341-346 (1968)

[?.) Hoare, C.A.R., Monitors: An Operating System Structuring Concept, CACM, vol.17, no.10, PP547-557 (1974

13] Hoare, C.A.R., Communicating Sequential Processes, CACM, vol.21, no.8, pp666-677 (1978)

[4] United States Dept. *of* Defence, Reference Manual for the Ada Programming Language (1980)

[5] Kanayama, Y., J. Iijima, II. Watarai and K. Ohkawa, A Self-Contained Robot "Yamabico, Proc. *of* 3rd UJCC, *pp246-250* (1978)

[6] Iijima, J., Y. Kanayama and S. Yuta, Locomotion Control System for Mobile Robots, Proc. of 7th IJCA1 pp779-784 (1981)

[7] Iijima, J., S. Yuta and Y. Kanayama, Elementary Functions of a Self-Contained Robot "Yamabico 3.1", Proc. of 11th ISIR, pp211-2I8 (1981)

[8] Motorola Inc., MC6809-MC6809E Microprocessor Programming Manual (1981)

[9] Kanayama, Y., RCS User's Manual, University of Tsukuba. (1983)

[10] Microware Systems Corporation, 0S-9 Level One Version 1.1 User's Guide (1981)

[II] Microware Systems Corporation, 0S-9 Level One Operating System VI.1 System Programmer's Manual (1981)

[12] Spier, M.J., and E.I. Organick, The Multics Interprocess Communication Facility, Proc. 2nd ACM Symposium on Operating systems Principles, pp83-91 (1969)

[13] Brinch Hansen, P., Operating System Principies, Prentice-Hall (1973)

[14] Kanayama, Y., S. Yuta, and J. Iijima, A Mobile Robot with Sonic Sensors and its Understanding of a Simple World, Report of Institute of Information Sciences and Electronics, University of Tsukuba (1981)

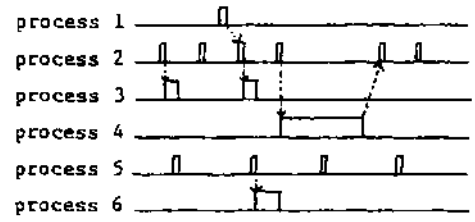

Figure 1 Execution of Concurrent Processes

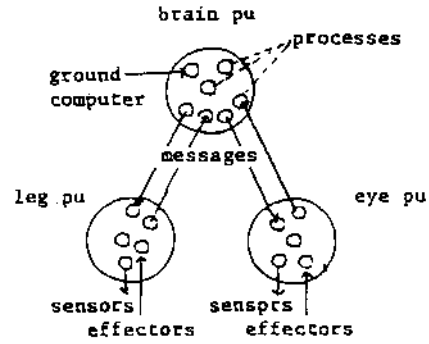

Figure 2

oncurrent Processes in a Multiprocessor System

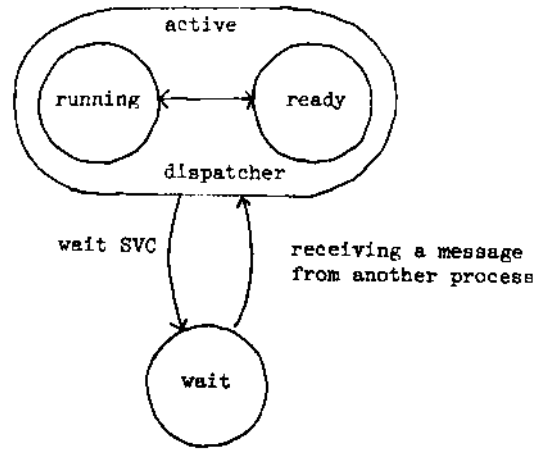

Figure 4 Transition between States

Figure 3 Yamabico 9

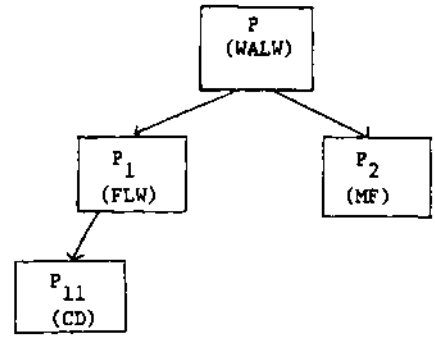

Figure 5 Process Family

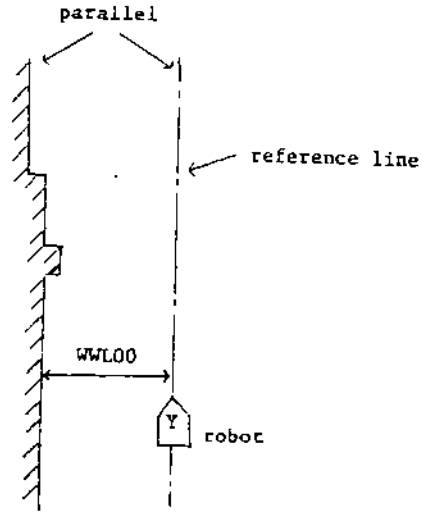

Figure 7 Follow Left Walls

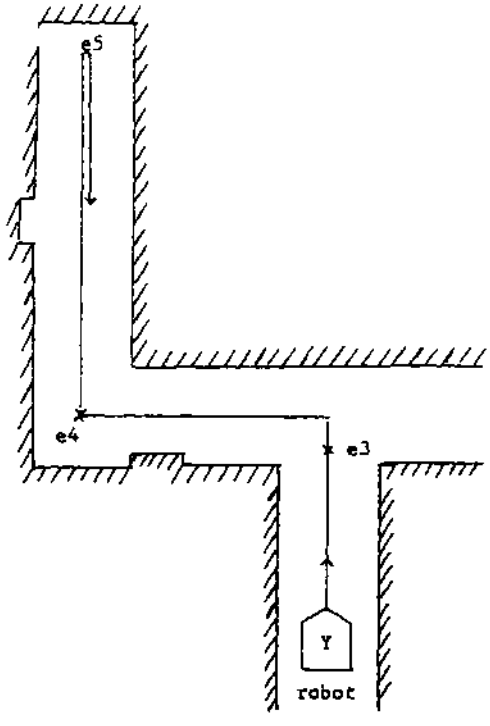

Figure 6 Walk Along Left Walls

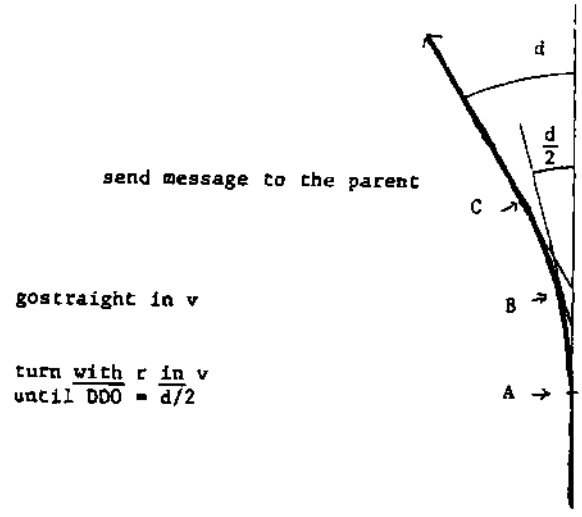

Figure 8 Change Direction