



(22) Date de dépôt/Filing Date: 1998/09/23

(41) Mise à la disp. pub./Open to Public Insp.: 1999/03/30

(45) Date de délivrance/Issue Date: 2012/06/12

(30) Priorité/Priority: 1997/09/30 (FR97/12146)

(51) Cl.Int./Int.Cl. *G06F 9/445* (2006.01),
G06F 9/50 (2006.01)

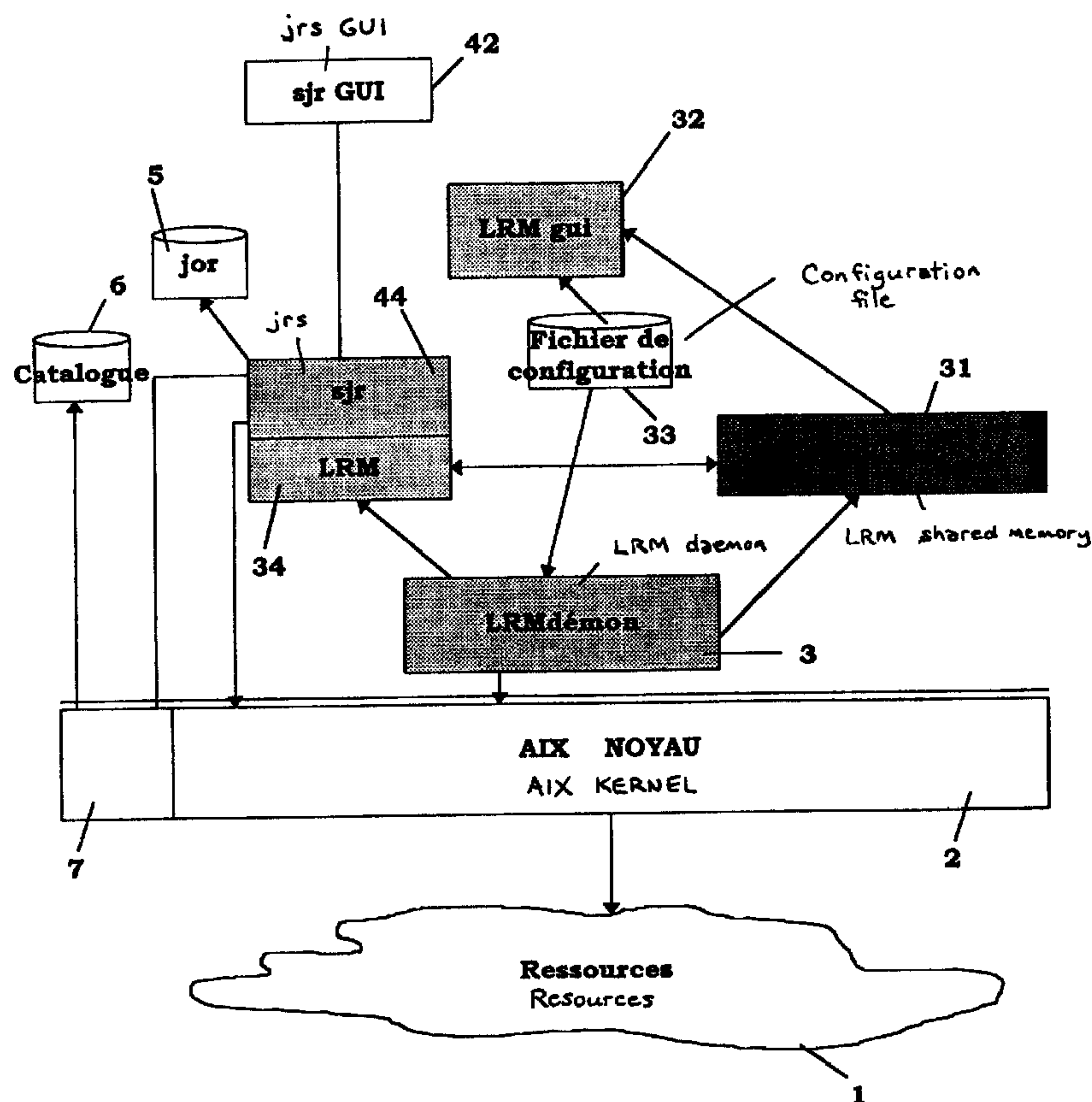
(72) Inventeurs/Inventors:
DURAND, DANIEL LUCIEN, FR;
SITBON, GERARD, FR;
URBAIN, FRANCOIS, FR

(73) Propriétaire/Owner:
BULL S.A., FR

(74) Agent: GOUDREAU GAGE DUBUC

(54) Titre : DISPOSITIF ET PROCESSUS GERANT L'EXECUTION D'UNE TACHE DANS UN SYSTEME DE
TRAITEMENT DE DONNEES

(54) Title: DEVICE AND PROCESS FOR HANDLING THE EXECUTION OF A JOB IN A DATA PROCESSING SYSTEM



(57) Abrégé/Abstract:

The present invention relates to a process and a device for handling the execution of a job in an open data processing system as a function of the resources. The process comprises the following steps: - determining the resources available in virtual memory, real

(57) **Abrégé(suite)/Abstract(continued):**

memory, temporary file space, central processing unit utilization time during the last time interval; - computing the amount of resources preallocated to other requests and not yet used; - comparing the amount of resources required for the execution of a job for which the request has been presented to the current amount of resources available minus the total amount of resources preallocated to other requests, in order to determine as a function of the result of this comparison the start, the deference or the denial of the start of the job requested.

ABSTRACT

The present invention relates to a process and a device for handling the execution of a job in an open data processing system as a function of the resources. The process comprises the following steps:

- determining the resources available in virtual memory, real memory, temporary file space, central processing unit utilization time during the last time interval;

- computing the amount of resources preallocated to other requests and not yet used;

- comparing the amount of resources required for the execution of a job for which the request has been presented to the current amount of resources available minus the total amount of resources preallocated to other requests, in order to determine as a function of the result of this comparison the start, the deference or the denial of the start of the job requested.

1 DEVICE AND PROCESS FOR HANDLING THE EXECUTION OF A JOB
2 IN A DATA PROCESSING SYSTEM
3

4 The present invention relates to a device and a process for
5 handling the execution of a job in a data processing system.

6 It applies to the field of data processing operations in an
7 industrial environment and particularly to data processing systems
8 of the "open" type running, for example, on software of the "UNIX™"
9 or "WINDOWS NT™" type. The servers that run on this type of software
10 are called "open" servers, and are highly valued at the present
11 time.

12 The drawback of these servers is that they were developed for
13 academic applications whose purposes were not the same as those for
14 industrial purposes. Thus, in a "UNIX™" operating system, the jobs,
15 or more precisely each of the processes comprising the jobs are
16 assigned an initial priority, either by the system or by the user
17 who enters these jobs. In order for a job to be considered by the
18 system to have a higher priority, the user must explicitly assign it
19 an initial priority, or manually modify its priority using an
20 administrative command. Thus, there is no control over or inter-
21 correlation between the priority levels assigned to one job or
22 another.

23 Likewise, in a standard "UNIX™" system, the processes can
24 start up without any limitation, even in heavily loaded systems,
25 without any attention to the number of resources currently being
26 used by the processes executed and without any attention to the
27 number of resources the new process will require. This results in
28 the following undesirable behavior: a group of processes is
29 executed, the critical processes as well as the non-critical
30 processes, and the critical processes, which must be completed at a
31 time desired by the user, have too few resources available to be
32 completed at the desired time. At the same time, non-critical
33 processes are using too many resources.

34 Another unacceptable behavior is that in which there are not
35 enough temporary files available and consequently, the write
36 operations into the files fail. Sometimes, the application_____

1 controls this error condition, sometimes it does not control it,
2 and this can generate a false behavior due to the fact that
3 incomplete files are generated in one step of a job and processed
4 in subsequent steps. Another detrimental behavior is that in
5 which there is not enough paging space and the system makes the
6 arbitrary decision to kill the most recent process without
7 considering its importance. This is unacceptable in an operating
8 environment.

9 That is why the primary object of the invention is to
10 propose a process for handling the execution of a job which makes
11 it possible to eliminate these drawbacks.

12 This object is achieved as a result of the fact that the
13 process for handling the execution of a job in an open data
14 processing system (for example of the "UNIX" type) as a function
15 of the resources is characterized in that it comprises the
16 following steps:

17 - determining the resources available in virtual memory,
18 real memory, temporary file space, central processing unit
19 utilization time during the last time interval;

20 - computing the amount of resources preallocated to other
21 requests and not yet used;

22 - comparing the amount of resources required for the
23 execution of a job for which the request has been presented to
24 the current amount of resources available minus the total amount
25 of resources preallocated to other requests, in order to
26 determine as a function of the result of this comparison the
27 start, the deference or the denial of the start of the job
28 requested.

29 According to another characteristic, the process includes a
30 step comprised of defining the amount of resources required for
31 the execution of a job by a specific command.

32 According to another characteristic, the process includes a
33 step comprised of activating or deactivating the mechanism for
34 determining whether there are enough system resources available.

35 Another object of the invention is to propose a device for

1 handling the execution of a job.

2 In accordance with this object, the device for handling the
3 execution of a job in an open operating system is characterized in
4 that it comprises:

5 - means for determining the resources available in virtual
6 memory, real memory, temporary file space, central processing unit
7 utilization time during a given time interval;

8 - means for computing the amount of resources preallocated to
9 other requests but not yet used;

10 - means for comparing the amount of resources required for the
11 execution of a job for which the request has been presented to the
12 current amount of resources available minus the total amount of
13 resources preallocated to other requests;

14 - and means for controlling the start, the deference or the
15 denial of the start of the job requested as a function of the result
16 supplied by the comparing means.

17 According to another characteristic, the device comprises a
18 means for defining the amount of resources required for the
19 execution of a job by a specific command.

20 According to another characteristic, the device comprises a
21 means for activating or deactivating the mechanism for determining
22 whether there are enough system resources available.

23 Other characteristics and advantages of the present invention
24 will emerge more clearly with the reading of the description below,
25 given in reference to the appended drawings, in which:

26 - Fig. 1 represents a schematic view of the data processing
27 system and of the software means associated with the data processing
28 system for enabling the process according to the invention to be
29 implemented;

30 - Fig. 2 represents an exemplary situation in the management
31 of the resources according to the invention.

32 Prior to the description of an exemplary embodiment of the
33 invention in "UNIX™" it is useful to recall the following
34 definitions.

35 Hereinafter, the term "process" or "job" designates any
36 execution of a program (and consequently, in particular, its

environment in the system) at a given instant, the program itself constituting an inert object stored on a disk in the form of an ordinary executable file. In "UNIX™" systems, the existence of two types of processes is known:

- the system processes, which are not attached to any terminal, which are created at the system startup or at dates set by the administrator of the system and which are only interrupted at the system shutdown. These include the "SWAPPER," a certain number of processes known as "daemons," such as for example the process which ensures the correct usage of the printer in "SPOOL," or the process "CRON," which makes it possible to start jobs at a given date.

- the processes started by a particular user from a given terminal at a given date. In particular, the fact that it is housed in a terminal under a given identification causes the start of a process corresponding to the execution of a file determined in advance for each user. This type of process nearly always corresponds to the execution of a command language interpreter (BOURNE shell or C-shell). For information's sake, let us also recall that the utilization of standard services by a user is carried out in a remote machine by means of commands which, in order to function in the remote machine, require the existence of specific processes known as "daemons."

The structure of the "UNIX™" system represented in Fig. 1 is constituted by the hardware resources of a data processing system 1, which communicates with a "UNIX™" kernel 2 that ensures the management of the memory and the low-level input-outputs, and the chaining of various "tasks" (jobs). Used around this kernel are one or more "shell"-type command language "interpreters," a messaging system and utility programs, which obviously include a C language compiler.

To meet the needs of multitask applications in an open environment, two "api" program modules were developed around these standard elements in "UNIX™" systems, one of which (JRS Job Reporting Service) provides reports on the execution of the jobs started by a user, while the other (LRM) makes it possible to

1 display the proper execution or the occurrence of failures and thus
2 to detect the causes. This first module will be called the "JRS"
3 (Job Reporting Service) and the other will be called the "LRM"
4 (Local Resource Management) module.

5 Added to this, for purposes of the invention, is a local
6 resource management daemon 3 which communicates with the kernel 2
7 and with an application program interface 34. This daemon 3 also
8 communicates with an LRM configuration file 33 stored in the hard
9 disk, and with an associated graphical user interface (LRM gui) 32.
10 The application program interface 34 communicates with the daemon 3
11 and with a shared memory 31. This shared memory is also in
12 communication with the graphical user interface 32 and the daemon 3.
13 For purposes of pre-allocation, the system incorporates "JRS"
14 application program interface 44 for reporting the execution of jobs
15 (Job Reporting Service). This application 44 communicates with a
16 catalogue "jor" which stores the job report in a distinct file 5 of
17 the hard disk. This application program interface 44, if it has been
18 activated and the job report files have been stored, communicates
19 with the local resource management daemon 3 so as to allow the pre-
20 allocation and dynamic control of resources. The interface 44
21 comprises the lines of code that are necessary to allow the
22 execution of the commands available to the user through this
23 interface. The commands available through this interface 44 are the
24 following:

25 - a command "JRS" which makes it possible to send a command to
26 the resource manager and to generate the execution of a report file.
27 This command, which appears in Appendix 1, comprises a plurality of
28 options which make it possible to determine the number of pages of
29 virtual memory, the quantity of temporary file space, the dimensions
30 defined by the users, the expected CPU time consumption, the maximum
31 elapsed time, the maximum CPU time, the maximum temporary file
32 space, the maximum number of pages of

virtual memory. This command appears in Appendix 1 of the specification with the meanings of these various parameters.

- a command "JRSjobstart" (int jobid ; int qflag ; struct JRS_resources *JRS_resources ;) in which the parameter jobid is the job identifier, and JRS-resources is the structure describing the resources required for the job. This can be 0 if the job does not require any pre-allocation of resources. The "qflag" parameter, when it is not indicates that the job will wait for the availability of the resources. This command verifies whether the job can start; if it cannot start, the system waits until the resources are available, which is the case when "qflag" is null. If "qflag" is not null, it causes an immediate return. If the job can start, a value 0 is returned; if not, when "qflag" is set, a value -1 is returned. Moreover, each major event of a job is stored as it occurs in a global file "logc" of the catalog "jor." This file can be displayed by means of the command "jrs log file," which allows the display of the parameters p, t, d, c, e, C, P, T defined below. In order to avoid having to explicitly define the parameters for each job entry, the application "JRS" 44 provides the capability to declare resources in a script. Thus, when the developer of an application has properly defined the resources needed for his job, he can insert the latter into the script. The job will be entered by the command "JRS" without any other parameter. The syntax for these command lines is the following:

#option-p "Number of pages of virtual memory"

#option-t "Amount of temporary file space"

#option-d "Function defined by the user"

#option-c "Expected CPU time"

#option-e "Maximum elapsed time"

#option-C "Limit of CPU time consumed"

#option-P "Maximum number of pages of virtual memory"

#option-T "Maximum temporary file space"

In case of a conflict, the value specified at the entry of the command is used as follows: if there is a line in the script such

1 as

2 #option-C 200

3 and if the script is entered with the following command line:

4 JRS-C 300 script,

5 then the value used for the CPU limit is 300.

6 With a system equipped in this way with the local resource
7 management module, the non-critical processes should alter the
8 continuation of the execution of the critical processes as little
9 as possible. Each type of process has a specific importance. More
10 or fewer resources should be allocated to it as a function of
11 this importance and as a function of the load on the system. For
12 this reason, the module "LRM" makes it possible to sort the
13 processes by dimension. A dimension is a set of commonly executed
14 processes which have the same importance from the point of view
15 of the local resource manager. By default, five dimensions are
16 provided: a first dimension "SYSTEM," a second dimension "BATCH,"
17 a third dimension "MISC," a fourth dimension DB (data base) and a
18 fifth dimension TP (transfer protocol). If the processes are
19 started by the normal users directly in the background, then they
20 belong to the dimension "MISC." If the processes are started via
21 a script, entered into the job management application without any
22 declaration of explicit dimensions, then they belong to the third
23 dimension "BATCH". If the processes do not belong to any other
24 dimension, then they belong to the dimension "SYSTEM." The
25 dimensions "DB" and "TP" are provided without any definitions of
26 processes and can be defined by the user. With the exception of
27 the dimension "SYSTEM," it is possible for each dimension to have
28 a relative weight, which is stored in a file. This relative
29 weight is used to manage the priority of the processes comprising
30 the dimension. The priority of all the processes belonging to a
31 dimension will vary accordingly. The dimension "SYSTEM" has no
32 relative weight and the priorities of the processes belonging to
33 this dimension are not modified by the local resource manager
34 (LRM). The processes controlled by the dimensions will always
35 have resources allocated to them as a function of their relative

1 weight, even if the system is heavily loaded. At the moment of
2 the startup of the local resource manager, or when a dimension is
3 created without specifying a relative weight, by default this
4 dimension will have a weight of -1, which means that it exists,
5 but that it is not controlled. A dimension can also be activated
6 or deactivated. If it is deactivated, the jobs entered that
7 belong to it will not be started, but the jobs in the process of
8 being executed in this dimension will continue to be executed.

9 When a process results from a job execution command "JRS,"
10 it belongs to the dimension specified, if a dimension has been
11 specified in the command "JRS." When the process has an ancestor
12 whose command name and user are associated with a dimension, the
13 process belongs to this dimension. During the run time of a
14 process, the relative weight will be used to manage and modify
15 the priority of the processes comprising the dimension. A
16 preallocated process either is not started by jrs or does not
17 comprise any preallocation parameters.

18 As a result of the resource management application, the user
19 as well as the resource manager will know for each resource at a
20 given time, how many resources are actually being used and how
21 many resources have been declared for each job. Thus, in the
22 example represented in Fig. 2, u^n represents the resources used
23 by the non-preallocated processes. R_a represents the amount of
24 the available resources reserved for the administrative and
25 system processes. U_1 , U_2 represent the resources used by the jobs
26 1 and 2. D_1 and D_2 represent the resources declared for the jobs
27 1 and 2. R_1 represents the difference between D_1 and U_1 . R_2
28 represents the difference between D_2 and U_2 . A_f represents the
29 resources considered available by the resource manager. If a job
30 is entered and declares an amount of necessary resources D_3 , it
31 will only start if A_f is greater than D_3 . This clearly shows the
32 advantage of the prior allocation of resources in allowing the
33 start of a job only when the minimum amount of necessary
34 resources is available.

35 The mechanism for preallocating virtual memory and temporary

1 file space is implemented by a command "JRS," which also
2 constitutes a process that contributes to the proper operation of
3 the system. When a job is entered, it can request via the command
4 JRS a certain amount of resources. The resource manager then
5 tries to ensure that there will be enough resources for the job.
6 The jobs can be entered with options, a first of which is to
7 supply a maximum elapsed time and a second of which is the
8 expected total central processing unit time for the job. When a
9 request for resources is examined, the following values are
10 considered:

- 11 - the amount of resources requested "r";
- 12 - the current amount of resources available "A";
- 13 - the resources already preallocated to other requests, but
14 not yet used "R";
- 15 - an initial amount of resources "Ra" declared at the start
16 by the user to the resource manager in order to aid in preventing
17 failures due to the uncontrolled allocation of resources.

18 The current amount of resources "A" is a piece of
19 information obtained periodically and updated periodically by the
20 "daemon" LRM by verifying the system information tables that have
21 been stored. The value of the resources already preallocated "R"
22 is also determined by the daemon "LRM," which gathers the current
23 utilizations of resources by the jobs. "Ra" is a set value,
24 supplied by the "daemon" LRM as being one of these arguments.

25 The request will be fulfilled if $r < A - (R + Ra)$. Upon each
26 request, the daemon performs this computation and fulfills or
27 does not fulfill the request as a function of the result of this
28 comparison. Once a request for resources corresponding to a job
29 start has been fulfilled, the job is executed, taking into
30 account the evolution of its priority.

31 The following appendices relate to the code parts that make
32 it possible to compute the total available virtual memory and
33 real memory. This computation is performed by the following code
34 part.

```

1  ...Computing Available and Total Virtual and Real Memory.
2      /*get_vmdata: Obtainment of Information on Available Real
3  and Virtual Memory */
4  get_vmdata(vm)
5      struct vmdata *vm
6  {
7      char    buff[1024];
8      int     bufsiz = 1024;
9      char    w1[20], w2[20], w3[20], w4[20];
10     while (1){
11         int     v1, v2;
12         if(!init_access_to_commands_done){
13 /*Initialization: Command vmstat is launched, and its output
14 caught*/
15             init_access_to_commands_done = 1;
16             init_access_to_commands();
17             fdvmcom = sag_popen_getline("vmstat 1", buff, &bufsiz,
18 2);
19         } else {
20             int cr;
21 /*A new line issued by command vmstat is read*/
22             cr = sag_popen_nextline(fdvmcom, buff, *bufsiz, 2);
23             if (cr <= 0){
24                 fdvmcom = sag_popen_getline("vmstat 1", buff,
25 *bufsiz, 2);
26             }
27         }
28 /*line is examined and information recovered*/
29         sscanf(buff, "%s %s %s %s", w1, w2, w3, w4);
30         if (!strcmp(w1, "kthr") && !strcmp(w2, "memory"))
31             continue;
32         if (!strcmp(w1, "-----") && !strcmp(w2, "-----"))
33             continue;
34         if (!strcmp(w1, "r") && !strcmp(w2, "b"))
35             continue;
36         sscanf(w3, "%d", &v1);
37         sscanf(w4, "%d", &v2);
38         vm->totrealmem = vmconstant.totrealmem;
39 /*System Call psdanger returns Total Virtual Memory and Available
40 Virtual Memory*/
41         vm->totvirtmem = psdanger(0) * 4;
42         vm->avrealmem = v2 * 4;
43         vm->avvirtmem = psdanger(-1) * 4;
44         return (0);
45     }
46 }

```

47 The process also comprises a step for computing the total
48 and available temporary file space. This computing step is
49 carried out by the following code part.

```

1  Computing Available and Total Temporary File Space
2
3  updtmpspace(avtmpspace, tottmpspace)
4      int      *avtmpspace;
5      int      *tottmpspace;
6  {
7      int      curnbtmpdev;
8      char      *ptdevlist;
9      int      i;
10     struct statfs  statfsbuf;
11
12     curnbtmpdev = enum_dev();
13     if (curnbtmpdev > nbtmpdev){
14         nbtmpdev = curnbtmpdev;
15         if (tmpdevlist)
16             (unsigned char *) free(tmpdevlist);
17         tmpdevlist = (char*) malloc(nbtmpdev *200);
18         tmpdevno = (int *) malloc(nbtmpdev *sizeof(int));
19     }
20     if (curnbtmpdev > 0){
21         get_dev(tmpdevlist, tmpdevno);
22         *avtmpspace = 0;
23         *tottmpspace = 0;
24         for (i = 0, ptdevlist = tmpdevlist; i < curnbtmpdev;
25 i++, ptdevlist += 200){
26             int      cr;
27             /*System Call ststfs provides information on the File System*/
28             cr = statfs(ptdevlist, &statfsbuf);
29             /*conversion into Kb: multiply by 4*/
30             *avtmpspace += 4 * statfsbuf.f_bfree;
31             *tottmpspace += 4 * statfsbuf.f_blocks;
32         }
33     }
34 }

```

Next, the central processing unit usage that occurred during the last time interval is determined, and this determination is made by means of the following code.

```

38     /*sysinfo and vminfo are AIX Kernel tables*/
39     /*The function get_sysinfo_and_vminfo reads these
40 tables from the Kernel Memory*/
41     get_sysinfo_andvminfo (&sysinfo, &vminfo);
42     tpgspgouts = (double) (vminfo.pgspgouts -
43 ovminfo.pgspgouts)/(double) Irmdinterval;
44     trunque = (double) (sysinfo.runque -
45 oysinfo.runque)/(double) Irmdinterval;
46     tswpque = (double) (sysinfo.swpque-

```

```

1      osysinfo.swpque)/(double) Irminterval;
2
3      tuser = (double) (sysinfo.cpu[CPU_USER])/((double)
4      Irminterval;
5      tsystem = (double) (sysinfo.cpu[CPU_KERNEL]-
6      osysinfo.cpu[CPU_KERNEL])/((double) Irminterval;
7      tidle = (double) (sysinfo.cpu[CPU_IDLE]-
8      osysinfo.cpu[CPU_IDLE])/((double)Irminterval;
9      twait = (double) (sysinfo.cpu[CPU_WAIT]-
10     osysinfo.cpu[CPU_WAIT])/((double) Irminterval;
11     tsum = tuser + tsystem + tidle + twait;
12

```

13 The process also comprises a step for determining the spaces
14 preallocated but not yet used by the jobs. This step is carried
15 out by the following code part.

```

16
17     Computing Prealloc space not used by jobs:
18
19     shm->system.preallocvirtmem = 0;
20     shm->system.prealloctmpspace = 0;
21     for (job = shm->first_job_started; job; job = job->next){
22         if (job->requiredvirtmem && (job->requiredvirtmem >
23 job->currentvirtmem)){
24             shm->system.preallocvirtmem += (job-
25 >requiredvirtmem - job->current virtmem);
26         }
27         if (job->requiredtmpspace && (job->requiredtmpspace >
28 job->currentusedtmpspace))
29             shm->system.prealloctmpspace += (job-
30 >requiredtmpspace - job->currentusedtmpspace);
31     }
32

```

33 The evaluation is carried out according to the formula $F < (R + Ra)$ and is constituted by the following code part.

```

34
35     Checking Resources for a job:
36
37     LRMresourcescheck (LRMresources)
38
39
40     struct LRMresources *LRMresources;
41     {
42         struct timeval tv;
43         struct timezone tz;
44         struct systementry system;

```

```

1      gettimeofday(&tv, &tz);
2      shmgetsystem(&system);
3      if (LRMresources->requiredtmpspace > (system.avtmpspace -
4 system.prealloctmpspace - system.reservedtmpspace)) ] 35
5          return (0);
6      if (LRMresources->requiredvirtmem > (system.avvirtmem -
7 system.preallocvirtmem - system.reservedvirtmem)) ] 36
8          return (0);
9      if (LRMresources->expectedtermination){
10         int      remains;
11         int      tl;
12         double    a, b;
13         remains=LRMresources->expectedtermination - tv.tv_sec;
14         LRMresources->neededratecpu = 100.0 * LRMresources->
15 expectedcpu / remains;
16         /*
17          * test here with the current available cpu rate
18          */
19         LRMdbgprintf(DBGALL, "Maximum Elapsed time specified\n");
20         LRMdbgprintf(DBGALL, "Remains: %d\n", remains);
21         LRMdbgprintf(DBGALL, "Needed Rate of CPU: %f\n",
22 LRMresources->neededratecpu);
23         LRMdbgprintf(DBGALL, "Extrapolated Available Rate
24 of CPU: %d\n", system.workingavcpu);
25         if (remains > 0 && LRMresources->neededratecpu <=
26 system.workingavcpu){
27             LRMdbgprintf(DBGALL, "Rate of CPU is good enough\n");
28             return (1);
29         } else {
30             LRMdbgprintf(DBGALL, "Rate of CPU is too low\n");
31             if (remains < 0 || LRMresources->neededratecpu >
32 100.0* system.nbprocessors){
33                 LRMdbgprintf(DBGALL, "There will not be enough
34 free CPU to complete the job in time\n");
35                 return (-1);
36             }
37             return (0);
38         }
39     } else
40         return (1);
41 }
42
43
44
45

```

46 In this code part, the portion 35 performs the evaluation of
47 the temporary space part and returns the value 0 if enough
48 temporary space is available. Then, the process continues with an
49 evaluation of the virtual memory part performed by the portion 36
which returns the value 0 if enough virtual memory space is

1 available. Finally, the system performs an evaluation of the
2 available CPU resources using the code part 38. This evaluation is
3 done after having determined the remaining resources (REMAINS) and
4 the necessary CPU rate (NEEDED RATE CPU). If the available CPU rate
5 is enough, the system returns the value 1, which allows the
6 execution of the request, whereas in the case where the system
7 returns the value -1, the request for the execution of the job is
8 deferred.

9 Thus, through the definition of the parameters necessary to
10 the execution of a job pre-entered into the command JRS by the user,
11 the latter can, using the mechanism described above, trigger the
12 execution of a job in the system, when the necessary resources for
13 its proper execution are available.

14 Any modification within reach of one skilled in the art is
15 also part of the spirit of the invention. Thus, the process and the
16 device described can be easily used and applied with another open
17 system such as, for example, "WINDOWS NT™".

APPENDIX 1

NAME: jrs command

jrs - submits a command to the resource manager and
generates an execution report in a file SYNOPTIQUE

jrs[-p <Number of virtual memory pages>
[-t <Amount of temporary file space>
[-q]
[-f]
[-l]
[-n]
[-o]
[-d <Dimension defined by user>]
[-c <Expected CPU time consumption>
-e <Maximum time elapsed>
[-C >Maximum CPU time>]
[-T <Maximum temporary file space>
[-P <Maximum number of pages of virtual memory>
[-r 0| 1| 2| 3]
[-v <environmentvariable> = <value>]...
[command args ...]

CLAIMS

1. A process for handling a request for the execution of a job, in an open data processing system, as a function of resources, comprising the following steps:

determining a current amount of resources available in virtual memory, real memory, temporary file space, and central processing unit utilization time during a given time interval;

computing an amount of resources preallocated to other requests and not yet used;

comparing an amount of resources required for the execution of the job, including an expected CPU time consumption, a maximum elapsed time, and a maximum CPU time consumed, to the current amount of resources available minus a total amount of preallocated resources including the amount of resources preallocated to other requests and not yet used as well as an amount of available resources reserved for administrative and system processes, in order to determine, as a function of a result of this comparison, one of: i) the start, ii) the deference and iii) the denial of the start of the job requested.

2. The process according to claim 1, including a step comprising defining the amount of resources required for the request for the execution of the job by a specific command.

3. The process according to any one of claims 1 and 2, including a step comprising one of: i) activating and ii) deactivating a mechanism for determining whether there are enough system resources available.

4. A device for handling a request for execution of a job, in an open data processing system, comprising:

means for determining a current amount of resources available in virtual memory, real memory, temporary file space,

and central processing unit utilization time during a given time interval;

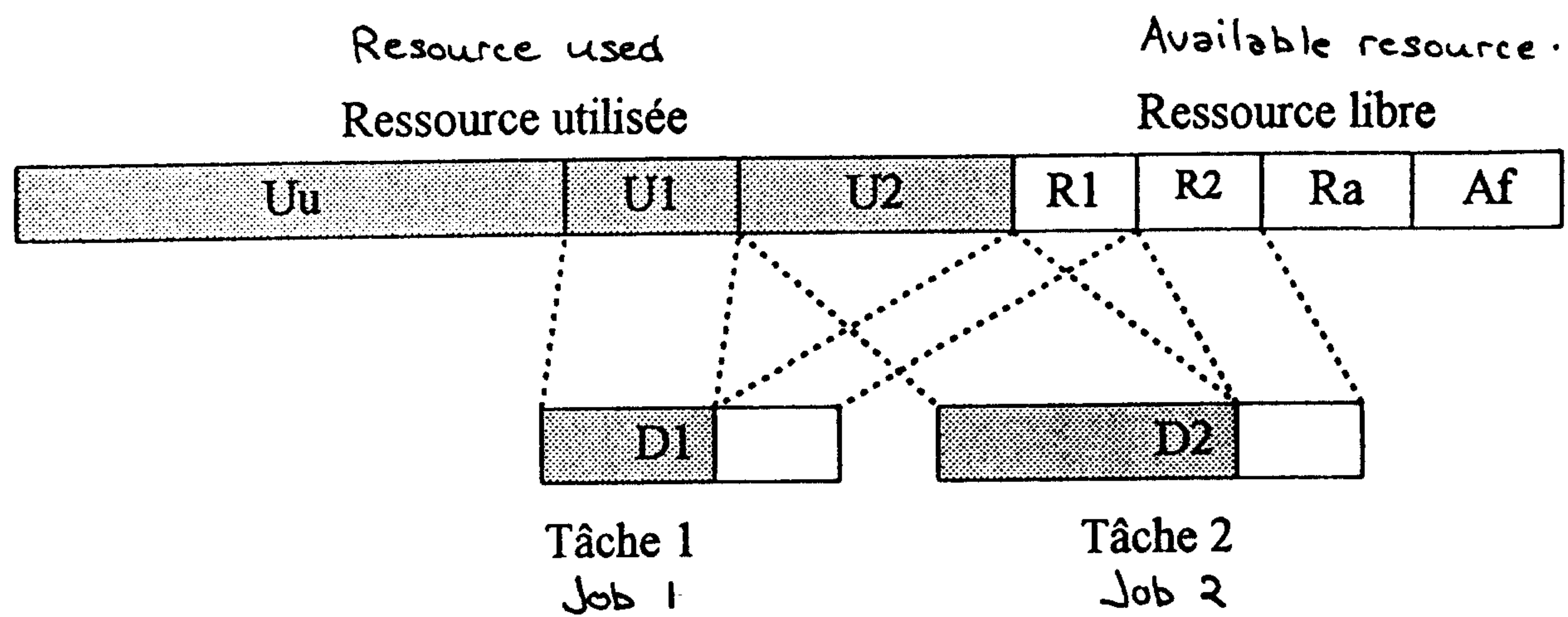
means for computing an amount of resources preallocated to other requests but not yet used;

means for comparing an amount of resources required for execution of the job, including an expected CPU time consumption, a maximum elapsed time, and a maximum CPU time, to the current amount of resources available, minus a total amount of the amount of resources preallocated to other requests and not yet used defined and an amount of available resources reserved for administrative and system processes; and

means for commanding one of: i) the start, ii) the deference and iii) the denial of the start of the job as a function of a result supplied by the means for comparing.

5. The device according to claim 4, comprising a means for defining the amount of resources required for the request for execution of the job by a specific command.

6. The device according to any one of claims 4 and 5, comprising a means for one of: i) activating and ii) deactivating a mechanism for determining whether there are enough system resources available.

**FIG. 2**

