BATCH SYSTEMS

The traditional batch operating system developed over many years into an extremely efficient means of using a large computer system. The important factor in measuring this efficiency is the amount of resource time spent on useful work (*not* just the active time !), and, given favourable circumstances, a good batch operating system could deliver up to 70% or 80% of the processor time for use on non-system tasks.

LINES OF DEVELOPMENT.

CONTINUOUS JOB STREAMS.

We saw earlier (*EFFICIENCY : GETTING THE MOST OUT OF THE MACHINE*) that the word "batch" was first associated with the operators' practice of collecting together in batches jobs with similar requirements. Jobs which required the same facilities could thus be processed together, and the operator could prepare for the batch by making ready the devices, programmes, and materials (tapes, utilities, printer paper, etc.) which it would require. In fact, while this is an efficient way for the operator to work, it is not so efficient for the processor, because it clusters jobs of similar characteristics. In order to make maximum use of the system's resources, we need a mixture of jobs of different characteristics, so that, ideally, there is always some programme ready to use a resource which becomes available.

In this respect, at least, the newer sort of batch system is quite different. To get a wide variety of jobs, it was sensible to accept all comers – the continuous job stream – rather than have the operator sort them out first. Of course, that doesn't mean that the jobs were executed in indiscriminate order; but the selection process is now the responsibility of the operating system, not the operator.

MULTIPLE JOB QUEUES.

How is the operating system to choose which job to execute next ? Evidently, if the aim is to select a mixture of jobs of complementary demands on the available resources, then the system must be able to find out what resources the jobs will require. There is no general way to automate this; you can't necessarily tell what a job will want by inspecting it, for, even if you could see all the job's demands on resources, you would have no guarantee that all the resource-using instructions would be executed, nor how many times they would be executed.

The system must therefore rely on the owners of the jobs to provide the information. It could in principle be done through special JCL statements which specified resources required; but it was more common to provide several different queues for jobs of different types. Provided that people entered their jobs into the queues which best matched their expected behaviour, the operating system could select a job which would be appropriate to achieving a balanced and high demand for all resources.

AUTOMATIC RESOURCE ALLOCATION.

Under this regime the operator can no longer take any part in allocating many resources to jobs, and most such activities must be performed by the operating system. In the cases of memory and disc space (and, of course, the processor), response to requests must be very fast if the system is not to be slowed down unacceptably; for slower devices, such as magnetic tape drives, a slower response is tolerable, but it is still important that waiting for manual operations should be cut down to the minimum.

RECONFIGURATION.

As an extreme case of automatic resource management, the system should be able to cope smoothly with hardware changes – such as taking a peripheral unit out of service for maintenance, or adding a new unit. This ideal is still not attained by many large operating systems; reconfiguration is only possible by stopping and restarting the whole system. Other systems are much more accommodating, and can cope with large changes – even up to adding or removing processors, provided that there's always at least one left.

RELIABILITY.

The system must not stop unexpectedly. Ever. This is not only a matter of immediate convenience. As the system becomes more complex, it necessarily carries more complex information structures in memory, and if these are lost, it might be impossible to keep the system in a self-consistent state. For example, a common consequence of such unexpected failures is the loss of records from files either because the current record was not written from memory to the disc when the failure occurred or because, while the record itself might have been written, the file system's information on the disc about the file had not yet been amended to show the presence of the new record.

AN EXAMPLE : THE BURROUGHS B6700 MCP.

The only reason for choosing this system is that it was used for a long time in Auckland University Computer Centre, and the information is ready to hand. We think it was a good system; but for present illustrative purposes pretty well any system would have done just as well.

THE SYSTEM STRUCTURE.

A sketch of the structure appears on page 32. The letters "B.C." which appear at the top right-hand corner are not part of the date, though the actual date which appears below might seem to you to be about as far back in time; they are the initials of the artist, Brent Callaghan, once an employee of the Auckland University Computer Centre, and last heard of at Sun Microsystems. Thanks, Brent. Notice that the work flows *upwards* in almost all parts of the diagram. Some explanatory notes on terms used in the diagram which are marked with an asterisk appear in the list below.

NOTES ON TERMS USED IN THE DIAGRAM.

TASKS and JOBS : a *task*, in WFL parlance, is a component of a *job*. Later in the course we shall call it a *process.* A *job* is roughly "the thing I want this deck of cards to do", and typically includes several tasks. A *task* is a single sequence of processor operations; a simple programme is usually executed as a single task, but tasks may initiate other tasks. Just to confuse the issue, there is a task for each WFL job; tasks can be nested. As the job task is executed, it starts other tasks corresponding to the different activities – executing programmes, operating system actions, and so on – of the job. These subtasks are commonly executed in sequence, but concurrent tasks can be set up if required.

AUTOPRINT : the print spooler – a programme which copies "printer backup" files from disc onto paper. Autoprint is a programme like any other, and has to take its turn for the processor. It is a bit cleverer than some spoolers in that it collects together all the printer files for a job and the job summary file – a record of what happened when the job was run – and prints them together.

CR10 : a card reader.

EI : a system control instruction entered by the operator (Emergency Interrupt – we don't know why) with which the operator opens or closes the queues. Normally the queues are left open and the system administers them automatically.

LP11, LP12 : two line printers.

MCP : the operating system. (Master Control Programme.)

NON-ACTIVE : a queue of jobs temporarily suspended by the system, usually awaiting some resource. They will be returned to the READY queue automatically when possible.

OK : a system control instruction entered by the operator to return a waiting task to the READY queue.

OVERALL MIXLIMIT : the *mix* is the set of jobs at present running; a *mixlimit* is a limit on the number of jobs which can be accepted at one time. The *overall mixlimit* is the maximum total number of jobs which can be accepted into the mix. The number can be changed by the operator.

PRINT QUEUE : a queue of files waiting to be printed.

PROCESSOR : the rather curious representation of the processor is a free interpretation of the main processor display panel. (Once upon a time, there really were computers which had front panels covered with flashing lights !)

QUEUE MIXLIMIT : the maximum number of active jobs from an individual queue. This can be set for each queue by the operator.

QUEUES : the thing which is actually queued is usually some sort of pointer – to a jobfile, in the early stages of the system, to an executing job (strictly, task) later on, and finally to a printer backup file.

READY QUEUE : tasks which are ready to run.

SCHEDULED TASKS : tasks which have been accepted from the queues and are waiting to be run.

SLOW READER : a card reader.

ST : a system control instruction entered by the operator to stop a task, and save it in the WAITING queue.

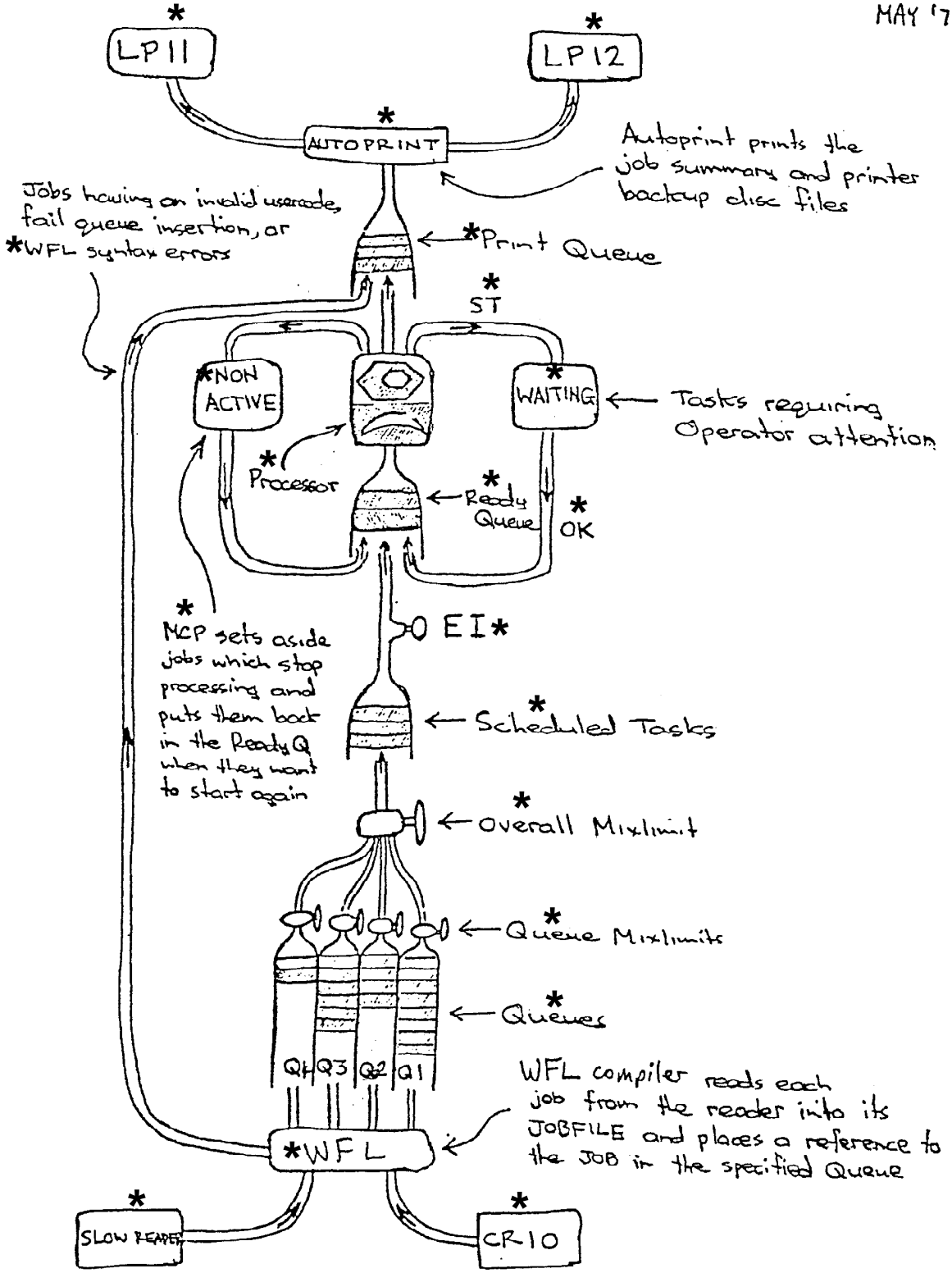
WAITING : a queue of tasks stopped by the operator.

WFL : WorkFlow Language – MCP's job control language. The MCP is unusual in having a true compiler for its job control language, rather than the more common interpreter. The WFL compiler acts as the system's input spooler.

WFL SYNTAX ERRORS : see WFL above.

B6700 WORKFLOW

B.C.
MAY '74



THE JOB QUEUES.

In a batch system like the MCP, once a job was entered its owner lost control. The only way in which you could influence your job in the system was by specifying its queue and certain resource limits; so these parameters assumed great importance. This extract from instructions issued by the computer centre[HIS4] shows how the queue system was organised.

Every job fed into the system is placed into the queue specified by the user's CLASS statement. If no queue is specified by the user, the job is placed in Q3 by default. The job queues are sometimes referred to as CLASS, e.g. a CLASS 3 job. For the purposes of the User Note, queue and class may be considered as being the same.

There are four queues which will cover the job requirements of most users, i.e. queues 1-4. Each queue has maximum limits on processor time, I/O time and number of lines printed. If a job exceeds those maximum limits it will be terminated by the system and the output will have the message "R-DSED" in the job summary. Users may set their own limits up to the maximum for the queue being used. Should a user submit a job with limits set that are greater than a queue permits, the job will be terminated by the system while still in the queue and the output for the job will have the message "DSED OUT OF QUEUE". When no limits are specified by a user the system will apply the default limits for the queue being used. A user may not have more than two jobs in a specified queue at the same time. The queue limits effective from 28 May 1979 (queue attributes or class factors are other terms sometimes used) are :

QUEUE LIMITS
(Default Queue is Q3)

		LIMIT	*Q U E U E*			
			1	2	3	4
a	MAXPROCTIME (in seconds)	default	120	20	20	120
		maximum	1,200	120	20	2,000
b	MAXIOTIME (in seconds)	default	120	20	20	120
		maximum	1,200	120	20	2,000
c	MAXLINES (in lines)	default	500	500	300	500
		maximum	2,400	1,200	500	3,600

Choosing the Queue to use

4. Characteristic of Job in Queues

a) Queue 3. This queue is to provide fast turnaround for short simple jobs such as those used in program development. To maintain turnaround the jobs should not require operator intervention, Queue 3 jobs are not permitted to use magnetic tapes or paper tape.

b) Queue 2. This queue is for reasonably short and simple jobs which may require up to two tapes (input or output).

c) Queue 1. Most jobs which do not meet the restriction of queues 2 and 3 should be submitted through Queue 1.

d) Queue 4. This queue is for jobs requiring a larger share of available System resources, for example jobs which exceed Ql limits or make use of large areas of disk or core. The nature of these jobs necessitates manual scheduling through the System.

e) Special Queue. Jobs which exceed even the requirements available through Q4 can be handed to the Secretary or the Operations Manager without a CLASS statement. They will be run when resources allow.

NOTE: Because of the effect some jobs have on the System, the Computer Centre reserves the right to direct Users to process jobs through a specified queue, notwithstanding that these jobs may be within the limits of a queue with faster turnaround.

5. Turnaround

The limits on the queues have been developed to enable the selection of fast, medium or slow turnaround on jobs. Fast turnaround has been provided to help Users doing program development or very small jobs. However, Users who break down larger jobs into smaller jobs to use a queue with faster turnaround will find that this practise is self-defeating. The turnaround for jobs in a particular queue depends on the volume of work in that queue. Details of the turnaround that may be expected from the queues under normal work-loads are:

a) Queue 3. Jobs in this queue should be turned around every half hour. Student Systems batch jobs do adversely affect this turnaround, especially in Terms II and III. At the worst, turnaround should not exceed one hour. Jobs put into this queue after 5 p.m. are unlikely to be run until the next day .

b) Queue 2. Normally an hourly turnaround will be maintained for jobs in this queue. However, large volumes of Q2 work frequently accumulate very rapidly, causing delays of 2-3 hours. Jobs entered into Q2 after 5 p.m. will not normally be processed until the next day.

c) Queue 1. For most of the year jobs entered into Queue 1 before 5 p.m. will be processed overnight. At other times, especially during Term III, the volume of work in the System could reduce turnaround to 48 hours.

d) Queue 4. Time available for processing this queue is limited and turnaround of 48 hours or longer will not be unusual. Users with two or more jobs in the queue will often be restricted to not more than one job being processed per night.

e) Special Queue. Turnaround cannot be estimated for this queue as jobs will be run only as resources allow.

REMARKS.

- "DS" means "discontinue". (From, one supposes, "di-scontinue", a most remarkable interpretation of the word !) "DSED" therefore means "discontinued", and "R-DSED" means, less obviously, "Resource-DSED", or discontinued for reasons connected with resource limits.

- Notice the evidence of war against unscrupulous people who try to get more than their share of the computing resource. "Users who break down larger jobs into smaller jobs" are mentioned; the limit of two jobs per queue per person was instituted as a way to control this phenomenon. (It was actually only applied to queues 1, 2, and 3, so the reference to "two or more jobs" in queue 4 isn't as silly as it sounds.)

REFERENCE.

HIS4 : *B6700 Job Handling* (User Note 1, Auckland University Computer Centre, September 1979)

QUESTIONS.

How must a job be represented in the operating system ? How would you represent multiple job queues ?

What does the operating system need to know in order to allocate resources to programmes which need them ? (Think of a convenient resource – say, memory – and pretend to be the operating system.)

How must an operating system handle reconfiguration without interfering with running programmes ?

The MCP, unusually, uses a compiler, which really generates executable code, to deal with its job control language. What advantages and disadvantages does a compiler have over the more usual interpreter for an operating system language ?

What does the operator do for a batch system ? How does that compare with the operator's job with a monitor system ?
