Group 3

| Họ và tên | English | VietNamese |
| --- | --- | --- |
| Nguyễn Đăng Nghĩa | 5.4.55  As we have seen, disks sometimes make errors. Good sectors can suddenly become bad  sectors. Whole drives can die unexpectedly. RAIDs protect against a few sectors going bad  or even a drive falling out. However, they do not protect against write errors laying down  bad data in the first place They also do not protect against crashes during writes corrupting  the original data without replacing them by newer data.  For some applications, it is essential that data never be lost or corrupted, even in the face of  disk and CPU errors. Ideally, a disk should simply work all the time with no errors.  Unfortunately, that is not achievable. What is achievable is a disk subsystem that has the  following property: when a write is issued to it, the disk either correctly writes the data or it  does nothing, leaving the existing data intact. Such as system is called stable storage and is  implemented in software (Lampson and Sturgis, 1979). Below we will describe a slight  variant of the original idea.  Before describing the algorithm, it is important to have a clear model of the possible errors.  The model assumes that when a disk writes a block (one or more sectors), either the write is  correct or it is incorrect and this error can be detected on a subsequent read by examining  the values of the ECC fields. In principle, guaranteed error detection is never possible  because with a, say, 16-byte ECC field guarding a 512-byte sector, there are 24096 data  values and only 2144 ECC values. Thus if a block is garbled during writing but the ECC is  not, there are billions upon billions of incorrect combinations that yield the same ECC. If  any of them occur, the error will not be detected. On the whole, the probability of random  data having the proper 16-byte FCC is about 2–144 , which is small enough that we will call  it zero, even though it is really not.  The model also assumes that a correctly written sector can spontaneously go bad and  become unreadable. However, the assumption is that such events are so rare that having the  same sector go bad on a second (independent) drive during a reasonable time interval (e.g.,  1 day) is small enough to ignore.  The model also assumes the CPU can fail, in which case it just stops. Any disk write in  progress at the moment of failure also stops, leading to incorrect data in one sector and an  incorrect ECC that can later be detected. Under all those conditions, stable storage can be  made 100% reliable in the sense of writes either working correctly or leaving the old data  in place. Of course, it does not protect against physical disasters, such as an earthquake  happening and the computer falling 100 meters into a fissure and landing in a pool of  boiling magma. It is tough to recover from this condition in software.  Stable storage uses a pair of identical disks with the corresponding blocks working together  to form one error-free block. In the absence of errors, the corresponding blocks on both  drives are the same. Either one can be read to get the same result. To achieve this goal, the  following three operations are defined:  1. Stable writes. A stable write consists of first writing the block on drive 1, then  reading it back to verify that it was written correctly. If it was not written  correctly, the write and reread are done again up to n times until they work.  After n consecutive failures, the block is remapped onto a spare and the  operation repeated until it succeeds, no matter how many spares have to be  tried. After the write to drive 1 has succeeded, the corresponding block on  drive 2 is written and reread, repeatedly if need be, until it, too, finally  succeeds. In the absence of CPU crashes, when a stable write completes, the  block has correctly been written onto both drives and verified on both of them.  2. Stable reads. A stable read first reads the block from drive 1. If this yields an  incorrect ECC, the read is tried again, up to n times. If all of these give bad  ECCs, the corresponding block is read from drive 2. Given the fact that a  successful stable write leaves two good copies of the block behind, and our  assumption that the probability of the same block spontaneously going bad on  both drives in a reasonable time interval is negligible, a stable read always  succeeds.  3. Crash recovery. After a crash, a recovery program scans both disks comparing  corresponding blocks. If a pair of blocks are both good and the same, nothing  is done. If one of them has an ECC error, the bad block is overwritten with the  corresponding good block. It a pair of blocks are both good but different, the  block from drive 1 is written onto drive 2.  In the absence of CPU crashes, this scheme always works because stable writes always  write two valid copies of every block and spontaneous errors are assumed never to occur on  both corresponding blocks at the same time. What about in the presence of CPU crashes  during stable writes? It depends on precisely when the crash occur. There are five  possibilities, as depicted in  . During  recovery, neither will be changed and the old value will continue to exist, which is allowed.  In Fig. 5-30(b), the CPU crashes during the write to drive l, destroying the contents of the  block. However the recovery program detects this error and restores the block on drive 1  from drive 2. Thus the effect of the crash is wiped out and the old state is fully restored.  The point of no return has been passed here: the recovery program copies the block from  drive 1 to drive 2. The write succeeds.  Again, the final value of both blocks is the new one.  Various optimizations and improvements are possible to this scheme. For starters,  comparing all the blocks pairwise after a crash is doable, but expensive. A huge  improvement is to keep track of which block was being written during a stable write so that  only one block has to be checked during recovery. Some computers have a small amount of  nonvolatile RAM which is a special CMOS memory powered by a lithium battery. Such  batteries last for years, possibly even the whole life of the computer. Unlike main memory,  which is lost after a crash, nonvolatile RAM is not lost after a crash. The time of day is  normally kept here (and incremented by a special circuit), which is why computers still  know what time it is even after having been unplugged.  Suppose that a few bytes of nonvolatile RAM are available for operating system purposes.  The stable write can put the number of the block it is about to update in nonvolatile RAM  before starting the write. After successfully completing the stable write, the block number  in nonvolatile RAM is overwritten with an invalid block number, for example, –1. Under  these conditions, after a crash the recovery program can check the nonvolatile RAM to see  if a stable write happened to be in progress during the crash, and if so, which block was  being written when the crashed happened. The two copies of the block can then be checked  for correctness and consistency.  If nonvolatile RAM is not available, it can be simulated as follows. At the start of a stable  write, a fixed disk block on drive 1 is overwritten with the number of the block to be stably  written. This block is then read back to verify it. After getting it correct, the corresponding  block on drive 2 is written and verified. When the stable write completes correctly, both  blocks are overwritten with an invalid block number and verified. Again here, after a crash  it is easy to determine whether or not a stable write was in progress during the crash. Of  course, this technique requires eight extra disk operations to write a stable block, so it  should be used exceedingly sparingly.  One last point is worth making. We assumed that only one spontaneous decay of a good  block to a bad block happens per block pair per day. If enough days go by, the other one  might go bad too. Therefore, once a day a complete scan of both disks must be done  repairing any damage. That way, every morning both disks are always identical. Even if  both blocks in a pair go bad within a period of a few days, all errors are repaired correctly | 5.4.55  Như chúng ta đã thấy, đĩa đôi khi mắc lỗi. Các lĩnh vực tốt có thể đột nhiên trở thành xấu  ngành. Toàn bộ ổ đĩa có thể chết bất ngờ. RAID bảo vệ chống lại một số lĩnh vực bị hỏng  hoặc thậm chí là một ổ đĩa bị rơi ra ngoài. Tuy nhiên, chúng không bảo vệ chống lại lỗi ghi đặt xuống  dữ liệu xấu ngay từ đầu Chúng cũng không bảo vệ khỏi sự cố trong quá trình ghi làm hỏng  dữ liệu ban đầu mà không thay thế chúng bằng dữ liệu mới hơn.  Đối với một số ứng dụng, điều cần thiết là dữ liệu không bao giờ bị mất hoặc bị hỏng, ngay cả khi đối mặt với  lỗi đĩa và CPU. Lý tưởng nhất là một đĩa đơn giản là hoạt động mọi lúc mà không có lỗi.  Thật không may, điều đó là không thể đạt được. Điều có thể đạt được là một hệ thống con đĩa có  thuộc tính sau: khi ghi được cấp cho nó, đĩa sẽ ghi dữ liệu chính xác hoặc nó  không làm gì cả, giữ nguyên dữ liệu hiện có. Hệ thống như vậy được gọi là lưu trữ ổn định và là  thực hiện trong phần mềm (Lampson và Sturgis, 1979). Dưới đây chúng tôi sẽ mô tả một chút  biến thể của ý tưởng ban đầu.  Trước khi mô tả thuật toán, điều quan trọng là phải có một mô hình rõ ràng về các lỗi có thể xảy ra.  Mô hình giả định rằng khi một đĩa ghi một khối (một hoặc nhiều cung), hoặc quá trình ghi là  đúng hay sai và lỗi này có thể được phát hiện trong lần đọc tiếp theo bằng cách kiểm tra  giá trị của các trường ECC. Về nguyên tắc, không bao giờ có thể phát hiện lỗi được đảm bảo  bởi vì với một trường ECC 16 byte bảo vệ một cung 512 byte, có 24096 dữ liệu  giá trị và chỉ 2144 giá trị ECC. Do đó, nếu một khối bị cắt xén trong khi ghi nhưng ECC vẫn  không, có hàng tỷ trên hàng tỷ kết hợp không chính xác mang lại cùng một ECC. Nếu như  bất kỳ trong số chúng xảy ra, lỗi sẽ không được phát hiện. Nhìn chung, xác suất ngẫu nhiên  dữ liệu có FCC 16 byte thích hợp là khoảng 2–144 , đủ nhỏ để chúng tôi gọi  nó bằng không, mặc dù nó thực sự không phải vậy.  Mô hình này cũng giả định rằng một khu vực được viết chính xác có thể bị lỗi một cách tự nhiên và  trở nên không thể đọc được. Tuy nhiên, giả định là những sự kiện như vậy hiếm đến mức có  cùng một khu vực bị lỗi trên ổ đĩa thứ hai (độc lập) trong một khoảng thời gian hợp lý (ví dụ:  1 ngày) đủ nhỏ để bỏ qua.  Mô hình cũng giả định rằng CPU có thể bị lỗi, trong trường hợp đó, nó chỉ dừng lại. Bất kỳ đĩa nào ghi vào  tiến trình tại thời điểm thất bại cũng dừng lại, dẫn đến dữ liệu không chính xác trong một lĩnh vực và một  ECC không chính xác mà sau đó có thể được phát hiện. Trong tất cả các điều kiện đó, lưu trữ ổn định có thể được  đáng tin cậy 100% theo nghĩa ghi hoạt động chính xác hoặc để lại dữ liệu cũ  tại chỗ. Tất nhiên, nó không bảo vệ chống lại các thảm họa vật lý, chẳng hạn như động đất  xảy ra và máy tính rơi xuống một khe nứt ở độ cao 100 mét và hạ cánh xuống một vũng nước  magma sôi. Rất khó để phục hồi từ tình trạng này trong phần mềm.  Bộ lưu trữ ổn định sử dụng một cặp đĩa giống hệt nhau với các khối tương ứng hoạt động cùng nhau  để tạo thành một khối không có lỗi. Trong trường hợp không có lỗi, các khối tương ứng trên cả hai  ổ đĩa là như nhau. Một trong hai có thể được đọc để có được kết quả tương tự. Để đạt được mục tiêu này, các  ba hoạt động sau đây được xác định:  1. Viết ổn định. Quá trình ghi ổn định bao gồm việc ghi khối đầu tiên trên ổ đĩa 1, sau đó  đọc lại để xác minh rằng nó đã được viết chính xác. Nếu nó không được viết  một cách chính xác, việc ghi và đọc lại được thực hiện lại tới n lần cho đến khi chúng hoạt động.  Sau n lần thất bại liên tiếp, khối này được ánh xạ lại vào một khối dự phòng và  hoạt động được lặp lại cho đến khi nó thành công, bất kể có bao nhiêu phụ tùng phải được  đã thử. Sau khi ghi vào ổ đĩa 1 thành công, khối tương ứng trên  ổ đĩa 2 được ghi và đọc lại, lặp đi lặp lại nếu cần, cho đến khi cuối cùng nó cũng vậy  thành công. Trong trường hợp không có sự cố CPU, khi quá trình ghi ổn định hoàn tất,  khối đã được ghi chính xác vào cả hai ổ đĩa và được xác minh trên cả hai ổ đĩa.  2. Đọc ổn định. Đọc ổn định đầu tiên đọc khối từ ổ đĩa 1. Nếu điều này mang lại  ECC không chính xác, quá trình đọc được thử lại, tối đa n lần. Nếu tất cả những điều này cho xấu  ECC, khối tương ứng được đọc từ ổ đĩa 2. Với thực tế là một  ghi ổn định thành công để lại hai bản sao tốt của khối và  giả định rằng xác suất của cùng một khối sẽ xấu đi một cách tự nhiên  cả hai ổ trong một khoảng thời gian hợp lý là không đáng kể, đọc ổn định luôn  thành công.  3. Phục hồi sự cố. Sau sự cố, chương trình khôi phục sẽ quét cả hai đĩa để so sánh  các khối tương ứng. Nếu một cặp khối đều tốt và giống nhau, không có gì  hoan thanh. Nếu một trong số chúng có lỗi ECC, khối xấu sẽ được ghi đè bằng  khối tốt tương ứng. Đó là một cặp khối đều tốt nhưng khác nhau,  khối từ ổ đĩa 1 được ghi vào ổ đĩa 2.  Trong trường hợp không có sự cố CPU, sơ đồ này luôn hoạt động vì luôn ghi ổn định  ghi hai bản sao hợp lệ của mỗi khối và các lỗi tự phát được cho là không bao giờ xảy ra trên  cả hai khối tương ứng cùng một lúc. Còn khi có sự cố CPU thì sao  trong quá trình ghi ổn định? Nó phụ thuộc vào chính xác khi sự cố xảy ra. Có năm  khả năng, như được mô tả trong.  . Trong lúc  phục hồi, sẽ không bị thay đổi và giá trị cũ sẽ tiếp tục tồn tại, điều này được cho phép.  Trong Hình 5-30(b), CPU gặp sự cố trong quá trình ghi vào ổ đĩa l, phá hủy nội dung của ổ đĩa l.  khối. Tuy nhiên, chương trình khôi phục phát hiện lỗi này và khôi phục khối trên ổ đĩa 1  từ ổ đĩa 2. Do đó, ảnh hưởng của sự cố bị xóa và trạng thái cũ được khôi phục hoàn toàn.  Điểm không quay lại đã được thông qua ở đây: chương trình khôi phục sao chép khối từ  ổ đĩa 1 sang ổ đĩa 2. Việc ghi thành công.  Một lần nữa, giá trị cuối cùng của cả hai khối là khối mới.  Đề án này có thể tối ưu hóa và cải tiến khác nhau. Cho người mới bắt đầu,  so sánh tất cả các khối theo cặp sau sự cố là có thể thực hiện được, nhưng tốn kém. Một khổng lồ  cải tiến là theo dõi khối nào đã được ghi trong quá trình ghi ổn định để  chỉ có một khối phải được kiểm tra trong quá trình khôi phục. Một số máy tính có một lượng nhỏ  RAM cố định là bộ nhớ CMOS đặc biệt chạy bằng pin lithium. Như là  pin kéo dài trong nhiều năm, thậm chí có thể là toàn bộ tuổi thọ của máy tính. Không giống như bộ nhớ chính,  bị mất sau sự cố, RAM cố định không bị mất sau sự cố. Thời gian trong ngày là  thường được giữ ở đây (và được tăng lên bởi một mạch đặc biệt), đó là lý do tại sao máy tính vẫn  biết bây giờ là mấy giờ ngay cả sau khi đã rút phích cắm.  Giả sử rằng một vài byte RAM cố định có sẵn cho các mục đích của hệ điều hành.  Quá trình ghi ổn định có thể đặt số khối mà nó sắp cập nhật vào RAM cố định  trước khi bắt đầu viết. Sau khi hoàn thành ghi ổn định thành công, số khối  trong RAM cố định được ghi đè bằng số khối không hợp lệ, ví dụ: –1. Dưới  những điều kiện này, sau khi gặp sự cố, chương trình khôi phục có thể kiểm tra RAM cố định để xem  nếu quá trình ghi ổn định xảy ra trong quá trình xảy ra sự cố và nếu có thì khối nào  được viết khi sự cố xảy ra. Hai bản sao của khối sau đó có thể được kiểm tra  cho đúng đắn và thống nhất.  Nếu không có RAM cố định, nó có thể được mô phỏng như sau. Khi bắt đầu ổn định  ghi, một khối đĩa cố định trên ổ đĩa 1 được ghi đè bằng số khối ổn định  bằng văn bản. Khối này sau đó được đọc lại để xác minh nó. Sau khi hiểu đúng, tương ứng  khối trên ổ đĩa 2 được viết và xác minh. Khi quá trình ghi ổn định hoàn thành chính xác, cả hai  các khối được ghi đè bằng số khối không hợp lệ và được xác minh. Một lần nữa ở đây, sau một vụ tai nạn  thật dễ dàng để xác định liệu quá trình ghi ổn định có đang diễn ra trong quá trình xảy ra sự cố hay không. Của  Tất nhiên, kỹ thuật này yêu cầu tám thao tác đĩa bổ sung để ghi một khối ổn định, vì vậy nó  nên được sử dụng cực kỳ tiết kiệm.  Một điểm cuối cùng đáng làm. Chúng tôi giả định rằng chỉ có một sự phân rã tự phát của một  khối thành khối xấu xảy ra trên mỗi cặp khối mỗi ngày. Nếu đủ ngày trôi qua, ngày khác  cũng có thể trở nên tồi tệ. Do đó, mỗi ngày một lần phải quét toàn bộ cả hai đĩa  sửa chữa bất kỳ thiệt hại. Bằng cách đó, mỗi sáng cả hai đĩa luôn giống hệt nhau. Thậm chí nếu  cả hai khối trong một cặp đều bị hỏng trong khoảng thời gian vài ngày, tất cả các lỗi đều được sửa chữa chính xác |
| Manh Toan | **5.4.4**  Disk manufacturers are constantly pushing the limits of the technology by increasing linear bit densities. A track midway out on a 5.25-inch disk has a circumference of about 300 mm. If the track holds 300 sectors of 512 bytes, the linear recording density may be about 5000 bits/mm taking into account the fact that some space is lost to preambles, ECCs, and intersector gaps. Recording 5000 bits/mm requires an extremely uniform substrate and avery fine oxide coating. Unfortunately, it is not possible to manufacture a disk to such specifications without defects. As soon as manufacturing technology has improved to the point where it is possible to operate flawlessly at such densities, disk designers will go to higher densities to increase the capacity. Doing so will probably reintroducde defects.Manufacturing defects introduce bad sectors, that is, sectors that do not correctly read back the value just written to them. If the defect is very small, say, only a few bits, it is possible to use the bad sector and just let the ECC correct the errors every time. If the defect is bigger, the error cannot be masked. There are two general approaches to bad blocks; deal with them in the controller or deal with them in the operating system. In the former approach, before the disk is shipped from the factory, it is tested and a list of bad sectors is written onto the disk. For each bad sector, one of the spares is substituted for it. There are two ways to do this substitution. In Fig. 5-29(a), we see a single disk track with 30 data sectors and two spares. Sector 7 is defective. What the controller can do is remap one of the spares as sector 7 as shown in Fig. 5-29(b). The other way is to shift all the sectors up one, as shown in Fig. 5-29(c). In both cases the controller has to know which sector is which. It can keep track of this information through internal tables (one per track) or by rewriting the preambles to give the remapped sector numbers. If the preambles are rewritten, the method of Fig. 5-29(c) is more work (because 23 preambles must be rewritten) but ultimately gives better performance because an entire track can still be read in one rotation.  Errors can also develop during normal operation after the drive has been installed. The first  line of defense upon getting an error that the ECC cannot handle is to just try the read  again. Some read errors are transient, that is, are caused by specks of dust under the head  and will go away on a second attempt, if the controller notices that it is getting repeated  errors on a certain sector, it can switch to a spare before the sector has died completely. In  this way, no data are lost and the operating system and user do not even notice the problem.  Usually, the method of Fig. 5-29(b) has to be used since the other sectors might now  contain data. Using the method of Fig. 5-29(c) would require not only rewriting the  preambles, but copying all the data as well.  Earlier we said there were two general approaches to handling errors: handle them in the  controller or in the operating system. If the controller does not have the capability to  transparently remap sectors as we have discussed, the operating system must do the same  thing in software. This means that it must first acquire a list of bad sectors, either by  reading them from the disk, or simply testing the entire disk itself. Once it knows which  sectors are bad, it can build remapping tables. If the operating system wants to use the  approach of Fig. 5-29(c), it must shift the data in sectors 7 through 29 up one sector.  If the operating system is handling the remapping, it must make sure that bad sectors do not  occur in any files and also do not occur in the free list or bitmap. One way to do this is to  create a secret file consisting of all the bad sectors. If this file is not entered into the file  system, users will not accidentally read it (or worse yet, free it).  However, there is still another problem: backups. If the disk is backed up file by file, it is  important that the backup utility not try to copy the bad block file. To prevent this, the  operating system has to hide the bad block file so well that even a backup utility cannot  find it. If the disk is backed up sector by sector rather than file by file, it will be difficult, if  not impossible, to prevent read errors during backup. The only hope is that the backup  program has enough smarts to give up after 10 failed reads and continue with the next  sector.  Bad sectors are not the only source of errors. Seek errors caused by mechanical problems in  the arm also occur. The controller keeps track of the arm position internally. To perform a  seek, it issues a series of pulses to the arm motor, one pulse per cylinder, to move the arm  to the new cylinder. When the arm gets to its destination, the controller reads the actual  cylinder number from the preamble of the next sector. If the arm is in the wrong place, a  seek error has occurred.  Most hard disk controllers correct seek errors automatically, but most floppy controllers  (including the Pentium’s) just set an error bit and leave the rest to the driver. The driver  handles this error by issuing a recalibrate command, to move the arm as far out as it will go  and reset the controller’s internal idea of the current cylinder to 0. Usually this solves the  problem. If it does not, the drive must be repaired.  As we have seen, the controller is really a specialized little computer, complete with  software, variables, buffers, and occasionally, bugs. Sometimes an unusual sequence of  events such as an interrupt on one drive occurring simultaneously with a recalibrate  command for another drive will trigger a bug and cause the controller to go into a loop or  lose track of what it was doing. Controller designers usually plan for the worst and provide  a pin on the chip which, when asserted, forces the controller to forget whatever it was  doing and reset itself. If all else fails, the disk driver can set a bit to invoke this signal and  reset the controller. If that does not help, all the driver can do is print a message and give  up.  Recalibrating a disk makes a funny noise but otherwise normally is not disturbing.  However, there is one situation where recalibration is a serious problem: systems with realtime constraints. When a video is being played off a hard disk, or files from a hard disk are  being burned onto a CD-ROM, it is essential that the bits arrive from the hard disk at a  uniform rate. Under these circumstances, recalibrations insert gaps into the bit stream and  are therefore unacceptable. Special drives, called AV disks (Audio Visual disks) , which  never recalibrate are available for such applications. | Các nhà sản xuất đĩa liên tục đẩy các giới hạn của công nghệ bằng cách tăng mật độ bit tuyến tính. Một rãnh ở giữa trên đĩa 5,25 inch có chu vi khoảng 300 mm. Nếu rãnh chứa 300 cung 512 byte, thì mật độ ghi tuyến tính có thể vào khoảng 5000 bit/mm có tính đến thực tế là một số khoảng trống bị mất cho phần mở đầu, ECC và các khoảng trống giao nhau. Ghi 5000 bit/mm yêu cầu chất nền cực kỳ đồng đều và lớp phủ oxit rất mịn. Thật không may, không thể sản xuất một đĩa theo các thông số kỹ thuật như vậy mà không có khuyết tật. Ngay khi công nghệ sản xuất đã được cải thiện đến mức có thể hoạt động hoàn hảo ở mật độ như vậy, các nhà thiết kế đĩa sẽ chuyển sang mật độ cao hơn để tăng dung lượng. Làm như vậy có thể sẽ đưa lại các lỗi. Các lỗi sản xuất dẫn đến các thành phần xấu, tức là các thành phần không đọc lại chính xác giá trị vừa được ghi cho chúng. Nếu lỗi rất nhỏ, chẳng hạn như chỉ một vài bit, thì có thể sử dụng khu vực xấu và chỉ cần để ECC sửa lỗi mỗi lần. Nếu lỗi lớn hơn, lỗi không thể được che giấu. Có hai cách tiếp cận chung đối với các khối xấu; xử lý chúng trong bộ điều khiển hoặc xử lý chúng trong hệ điều hành. Theo cách tiếp cận cũ, trước khi đĩa được vận chuyển từ nhà máy, nó sẽ được kiểm tra và một danh sách các thành phần xấu được ghi vào đĩa. Đối với mỗi khu vực xấu, một trong những phụ tùng được thay thế cho nó. Có hai cách để thực hiện việc thay thế này. Trong Hình 5-29(a), chúng ta thấy một rãnh đĩa đơn với 30 cung dữ liệu và hai rãnh dự phòng. Sector 7 bị lỗi. Những gì bộ điều khiển có thể làm là ánh xạ lại một trong các dự phòng thành khu vực 7 như trong Hình 5-29(b). Một cách khác là chuyển tất cả các cung lên một, như trong Hình 5-29(c). Trong cả hai trường hợp, bộ điều khiển phải biết khu vực nào là khu vực nào. Nó có thể theo dõi thông tin này thông qua các bảng nội bộ (một bảng trên mỗi rãnh) hoặc bằng cách viết lại phần mở đầu để cung cấp số khu vực được ánh xạ lại. Nếu phần mở đầu được viết lại, phương pháp của Hình 5-29(c) hiệu quả hơn (vì 23 phần mở đầu phải được viết lại) nhưng cuối cùng mang lại hiệu suất tốt hơn vì toàn bộ bản nhạc vẫn có thể được đọc trong một lần quay.Lỗi cũng có thể phát triển trong quá trình hoạt động bình thường sau khi ổ đĩa đã được cài đặt. Đầu tiên  hàng phòng thủ khi gặp lỗi mà ECC không thể xử lý là chỉ thử đọc  lại. Một số lỗi đọc là thoáng qua, nghĩa là, được gây ra bởi các đốm bụi dưới đầu  và sẽ biến mất trong nỗ lực thứ hai, nếu bộ điều khiển thông báo rằng nó đang được lặp lại  Lỗi trên một lĩnh vực nhất định, nó có thể chuyển sang phụ tùng trước khi khu vực đã chết hoàn toàn. TRONG  Bằng cách này, không có dữ liệu nào bị mất và hệ điều hành và người dùng thậm chí không nhận thấy vấn đề.  Thông thường, phương pháp của Hình 5-29 (b) phải được sử dụng vì các lĩnh vực khác có thể  chứa dữ liệu. Sử dụng phương pháp của Hình 5-29 (c) sẽ không chỉ yêu cầu viết lại  Mở đầu, nhưng sao chép tất cả các dữ liệu là tốt.  Trước đó chúng tôi đã nói có hai cách tiếp cận chung để xử lý lỗi: xử lý chúng trong  bộ điều khiển hoặc trong hệ điều hành. Nếu bộ điều khiển không có khả năng  các lĩnh vực sao chép trong suốt như chúng ta đã thảo luận, hệ điều hành phải làm như vậy  điều trong phần mềm. Điều này có nghĩa là trước tiên nó phải có được một danh sách các lĩnh vực xấu, bằng cách  Đọc chúng từ đĩa, hoặc chỉ đơn giản là kiểm tra toàn bộ đĩa. Một khi nó biết cái nào  Các lĩnh vực là xấu, nó có thể xây dựng các bảng ánh xạ lại. Nếu hệ điều hành muốn sử dụng  Cách tiếp cận của Hình 5-29 (c), nó phải chuyển dữ liệu theo các lĩnh vực 7 đến 29 lên một khu vực.  Nếu hệ điều hành đang xử lý ánh xạ lại, nó phải đảm bảo rằng các lĩnh vực xấu không  xảy ra trong bất kỳ tệp nào và cũng không xảy ra trong danh sách hoặc bitmap miễn phí. Một cách để làm điều này là  Tạo một tập tin bí mật bao gồm tất cả các lĩnh vực xấu. Nếu tệp này không được nhập vào tệp  Hệ thống, người dùng sẽ không vô tình đọc nó (hoặc tệ hơn là miễn phí).  Tuy nhiên, vẫn còn một vấn đề khác: sao lưu. Nếu đĩa được sao lưu tệp theo tệp, nó là  quan trọng là tiện ích sao lưu không cố gắng sao chép tệp khối xấu. Để ngăn chặn điều này,  hệ điều hành phải ẩn tệp khối xấu đến mức ngay cả một tiện ích sao lưu cũng không thể  tìm nó. Nếu đĩa được hỗ trợ theo ngành theo ngành chứ không phải tệp, thì sẽ rất khó khăn, nếu  Không phải là không thể, để ngăn ngừa lỗi đọc trong quá trình sao lưu. Hy vọng duy nhất là bản sao lưu  Chương trình có đủ thông minh để từ bỏ sau 10 lần đọc thất bại và tiếp tục với chương trình tiếp theo  khu vực.  Các lĩnh vực xấu không phải là nguồn lỗi duy nhất. Tìm kiếm lỗi do các vấn đề cơ học trong  Cánh tay cũng xảy ra. Bộ điều khiển theo dõi vị trí cánh tay trong nội bộ. Để thực hiện a  Tìm kiếm, nó đưa ra một loạt các xung cho động cơ cánh tay, một xung trên mỗi xi -lanh, để di chuyển cánh tay  đến xi lanh mới. Khi cánh tay đến đích, bộ điều khiển sẽ đọc thực tế  Số xi lanh từ phần mở đầu của khu vực tiếp theo. Nếu cánh tay ở sai vị trí, một  Tìm kiếm lỗi đã xảy ra. Bộ điều khiển đĩa cứng tự động tìm kiếm lỗi chính xác, nhưng hầu hết các bộ điều khiển mềm  (bao gồm cả pentium,) chỉ cần đặt một bit lỗi và để phần còn lại cho người lái xe. Người lái xe  Xử lý lỗi này bằng cách ban hành lệnh hiệu chỉnh lại, để di chuyển cánh tay ra xa như nó sẽ đi  và đặt lại ý tưởng nội bộ của bộ điều khiển của xi lanh hiện tại thành 0. Thông thường điều này giải quyết  vấn đề. Nếu không, ổ đĩa phải được sửa chữa.  Như chúng ta đã thấy, bộ điều khiển thực sự là một máy tính nhỏ chuyên dụng, hoàn chỉnh với  Phần mềm, biến, bộ đệm, và đôi khi, lỗi. Đôi khi một chuỗi khác thường của  các sự kiện như ngắt trên một ổ xảy ra đồng thời với một hiệu chỉnh lại  lệnh cho một ổ đĩa khác sẽ kích hoạt một lỗi và khiến bộ điều khiển đi vào một vòng lặp hoặc  mất dấu vết của những gì nó đã làm. Các nhà thiết kế điều khiển thường lập kế hoạch cho điều tồi tệ nhất và cung cấp  một pin trên chip, khi được khẳng định, buộc bộ điều khiển phải quên bất cứ thứ gì  Làm và đặt lại chính nó. Nếu tất cả những thứ khác bị lỗi, trình điều khiển đĩa có thể đặt một chút để gọi tín hiệu này và  Đặt lại bộ điều khiển. Nếu điều đó không giúp được gì, tất cả người lái có thể làm là in một tin nhắn và đưa ra  hướng lên.  Hiệu chỉnh một đĩa làm cho một tiếng ồn vui nhộn nhưng nếu không thì thông thường không đáng lo ngại.  Tuy nhiên, có một tình huống trong đó hiệu chỉnh lại là một vấn đề nghiêm trọng: các hệ thống có các ràng buộc thời gian thực. Khi một video đang được phát một đĩa cứng hoặc các tệp từ đĩa cứng là  bị đốt cháy trên CD-ROM, điều cần thiết là các bit đến từ đĩa cứng tại một  Tỷ lệ thống nhất. Trong những trường hợp này, các hiệu chỉnh lại chèn khoảng trống vào luồng bit và  do đó là không thể chấp nhận được. Các ổ đĩa đặc biệt, được gọi là đĩa AV (đĩa trực quan âm thanh), mà  Không bao giờ hiệu chỉnh lại có sẵn cho các ứng dụng như vậy. |
| Vu Ngoc Hai | 5.6 CHARACTER-ORIENTED TERMINALS  Every general-purpose computer has at least one keyboard and one display (monitor or flat  screen) used to communicate with it. Although the keyboard and display on a personal  computer are technically separate devices, they work closely together. On mainframes,  there are frequently many remote users, each with a device containing a keyboard and an  attached display. These devices have historically been called terminals . We will continue  to use that term, even when discussing personal computers (mostly for lack of a better  term).  Terminals come in many forms. Three of the types most commonly encountered in practice  nowadays are  1. Standalone terminals with RS-232 serial interfaces for use on mainframes.  2. Personal computer displays with graphical user interfaces.  3. Network terminals.  Each type of terminal has its own ecological niche. In the following sections we will  describe each of these types in turn.  5.6.1 RS-232 Terminal Hardware  RS-232 terminals are hardware devices containing both a keyboard and a display and which  communicate using a serial interface, one bit at a time (see Fig. 5-34). These terminals use  a 9-pin or 25-pin connector, of which one pin is used for transmitting data, one pin is for  receiving data, and one pin is ground. The other pins are for various control functions, most  of which are not used. Lines in which characters are sent one bit at a time (as opposed to 8  bits in parallel the way printers are interfaced to PCs) are called serial lines . All modems  also use this interface. On UNIX, serial lines have names like /dev/tty1 and /dev/tty2 . On  Windows they have names like COM1 and COM2.  To send a character over a serial line to an RS-232 terminal or modem, the computer must  transmit it 1 bit at a time, prefixed by a start bit, and followed by 1 or 2 stop bits to delimit  the character. A parity bit which provides rudimentary error detection may also be inserted  preceding the stop bits, although this is commonly required only for communication with  mainframe systems.  Figure 5-34. An RS-232 terminal communicates with a computer over a communication  line, one bit at a time.  RS-232 terminals are still commonly used in the mainframe world to allow a remote user to  communicate with the mainframe, sometimes using a modem and a telephone line. They  are found in the airline, banking, and other industries. Even when they are replaced by  personal computers, the PC’s often simply emulate the old RS-232 terminals to avoid  having to change the mainframe software.  These terminals also used to dominate the minicomputer world. A great deal of software for  systems that grew up in this period are based on these terminals. For example, all UNIX  systems support this kind of device.  However, even more important, many current UNIX systems (and other systems) provide  the option of creating a window consisting of some number of lines of text. Many  programmers work almost exclusively in text mode in such windows, even on personal  computers or high-end workstations. These windows usually simulate some RS-232  terminal (or the ANSI standard for this type of terminal) so they can run the large existing  software base that was written for such terminals. In the course of the years, this software,  such as the vi and emacs editors, has become completely debugged and extremely stable,  properties programmers value highly.  The keyboard and terminal software for these terminal emulation windows is the same as  for the real terminals. Since these terminal emulators are in widespread use, the software is  still important, so we will describe it in the following two sections.  RS-232 terminals are character oriented. What this means is that the screen or window  displays a certain number of lines of text, each of a maximum size. A typical size is 25 lines  of 80 characters each. While a few special characters are sometimes supported, these  terminals (and the emulators) are basically text only.  Since both computers and terminals work internally with whole characters but must  communicate over a serial line a bit at a time, chips have been developed to do the  character-to-serial and serial-to-character conversions. They are called UARTs (Universal  Asynchronous Receiver Transmitters). UARTs are attached to the computer by plugging  RS-232 interface cards into the bus as illustrated in Fig. 5-34. On many computers, one or  two serial ports are built into the parent-board.  To display a character, the terminal driver writes the character to the interface card, where  it is buffered and then shifted out over the serial line one bit at a time by the UART. For  example, for an analog modem operating at 56,000 bps, it takes just over 179 sec to send a  character. As a result of this slow transmission rate, the driver generally outputs a character  to the RS-232 card and blocks, waiting for the interrupt generated by the interface when the  character has been transmitted and the UART is able to accept another cha  racter. The  UART can send and receive characters simultaneously. An interrupt is also generated when  a character is received, and usually a small number of input characters can be buffered. The  terminal driver must check a register when an interrupt is received to determine the cause  of the interrupt. Some interface cards have a CPU and memory and can handle multiple  lines, taking over much of the I/O load from the main CPU.  RS-232 terminals can be subdivided into three categories. The simplest ones are hardcopy  (i.e., printing) terminals. Characters typed on the keyboard are transmitted to the computer.  Characters sent by the computer are printed on the paper. These terminals are obsolete and  rarely seen any more except as low-end printers.  Dumb CRT terminals work the same way, only with a screen instead of paper. These are  often called “glass ttys” because they are functionally the same as hardcopy ttys. (The term  "tty" is an abbreviation for Teletype, a former company that pioneered in the computer  terminal business: “tty” has come to mean any terminal.) Glass ttys are also obsolete.  Intelligent CRT terminals are in fact miniature, specialized computers. They have a CPU  and memory and contain software, usually in ROM. From the operating system’s  viewpoint, the main difference between a glass tty and an intelligent terminal is that the  latter understands certain escape sequences. For example, by sending the ASCII ESC  character (0x1B), followed by various other characters, it may be possible to move the  cursor to any position on the screen, insert text in the middle of the screen, and so forth.  Intelligent terminals are the ones used in mainframe systems and are the ones emulated by  other operating systems. It is their software that we will discuss below.  5.6.2 Input Software  The keyboard and display are almost independent devices, so we will treat them separately  here. They are not quite independent, however, since typed characters generally are  displayed on the screen.  The basic job of the keyboard driver is to collect input from the keyboard and pass it to user  programs when they read from the terminal. Two possible philosophies can be adopted for  the driver. In the first one, the driver’s job is just to accept input and pass it upward  unmodified. A program reading from the terminal gets a raw sequence of ASCII codes.  (Giving user programs the key numbers is too primitive, as well as being highly machine  dependent.)  This philosophy is well suited to the needs of sophisticated screen editors such as emacs ,  which allow the user to bind an arbitrary action to any character or sequence of characters.  It does, however, mean that if the user types dste instead of date and then corrects the error  by typing three backspaces and ate , followed by a carriage return, the user program will be  given all 11 ASCII codes typed, as follows:  d s t e a t e CR  Not all programs want this much detail. Often they just want the corrected input, not the  exact sequence of how it was produced. This observation leads to the second philosophy:  the driver handles all the intraline editing, and just delivers corrected lines to the user  programs. The first philosophy is character-oriented; the second one is line oriented.  Originally they were referred to as raw mode and cooked mode , respectively. The POSIX  standard uses the less-picturesque term canonical mode to describe line oriented mode.  Noncanonical mode is equivalent to raw mode, although many details of terminal behavior  can be changed. POSIX-compatible systems provide several library functions that support  selecting either mode and changing many aspects of terminal configuration.  The first task of the keyboard driver is to collect characters. If every keystroke causes an  interrupt, the driver can acquire the character during the interrupt. If interrupts are turned  into messages by the low-level software, it is possible to put the newly acquired character  in the message. Alternatively, it can be put in a small buffer in memory and the message  used to tell the driver that something has arrived. The latter approach is actually safer if a  message can be sent only to a waiting process and there is some chance that the keyboard  driver might still be busy with the previous character.  If the terminal is in canonical (cooked) mode, characters must be stored until an entire line  has been accumulated, because the user may subsequently decide to erase part of it. Even if  the terminal is in raw mode, the program may not yet have requested input, so the  characters must be buffered to allow type ahead. (System designers who do not allow users  to type far ahead ought to be tarred and feathered, or worse yet, be forced to use their own  system.)  Two approaches to character buffering are common. In the first one, the driver contains a  central pool of buffers, each buffer holding perhaps 10 characters. Associated with each  terminal is a data structure, which contains, among other items, a point er to the chain of  buffers for input collected from that terminal. As more characters are typed, more buffers  are acquired and hung on the chain. When the characters are passed to a user program, the  buffers are removed and put back in the central pool.  The other approach is to do the buffering directly in the terminal data structure itself, with  no central pool of buffers. Since it is common for users to type a command that will take a  little while (say, recompiling and linking a large binary program) and then type a few lines  ahead, to be safe the driver should allocate something like 200 characters per terminal. In a  large-scale timesharing system with 100 terminals, allocating 20K all the time for type  ahead is clearly overkill, so a central buffer pool with space for perhaps 5K is probably  enough. On the other hand, a dedicated buffer per terminal makes the driver simpler (no  linked list management) and is to be preferred on personal computers with only one  keyboard. Figure 5-35 shows the difference between these two methods.  Although the keyboard and display are logically separate devices, many users have grown  accustomed to seeing the characters they have just typed appear on the screen. Some (older)  terminals oblige by automatically displaying (in hardware) whatever has just been typed,  which is not only a nuisance when passwords are being entered but greatly limits the  flexibility of sophisticated editors and other programs. Fortunately, with most terminals,  nothing is automatically displayed when a key is struck. It is entirely up to the software in  the computer to display the character, if desired. This process is called echoing .  Figure 5-35. (a) Central buffer pool. (b) Dedicated buffer for each terminal.  Echoing is complicated by the fact that a program may be writing to the screen while the  user is typing. At the very least, the keyboard driver has to figure out where to put the new  input without it being overwritten by program output.  Echoing also gets complicated when more than 80 characters have to be displayed on a  screen with 80-character lines (or some other number). Depending on the application,  wrapping around to the next line may be appropriate. Some drivers just truncate lines to 80  characters by throwing away all characters beyond column 80.  Another problem is tab handling. It is usually up to the driver to compute where the cursor  is currently located, taking into account both output from programs and output from  echoing, and compute the proper number of spaces to be echoed.  Now we come to the problem of device equivalence. Logically, at the end of a line of text,  one wants a carriage return, to move the cursor buck to column 1, and a linefeed, to  advance to the next line. Requiring users to type both at the end of each line would not sell  well (although some terminals have a key which generates both, with a 50 percent chance  of doing so in the order that the software wants them). It is up to the driver to convert  whatever comes in to the standard internal format used by the operating system.  If the standard form is just to store a linefeed (the UNIX convention), then carriage returns  should be turned into linefeeds. If the internal format is to store both (the Windows  convention), then the driver should generate a linefeed when it gets a carriage return and a  carriage return when it gets a linefeed. No matter what the internal convention, the terminal  may require both a linefeed and a carriage return to be echoed in order to get the screen  updated properly. Since a large computer may well have a wide variety of different  terminals connected to it, it is up to the keyboard driver to get all the different carriage  return/linefeed combinations converted to the internal system standard and arrange for all  echoing to be done right. When operating in canonical mode, a number of input characters  have special meanings. Figure 5-36 shows all of the special characters required by POSIX.  The defaults are all control characters that should not conflict with text input or codes used  by programs, but all except the last two can be changed under program control.  Character  POSIX name  Comment  CTRL-H  ERASE  Backspace one character  CTRL-U  KILL  Erase entire line being typed  CTRL-V  LNEXT  Interpret next character literally  CTRL-S  STOP  Stop output  CTRL-Q  START  Start output  DEL  INTR  Interrupt process (SIGINT)  CTRL-\  QUIT  Force core dump (SIGQUIT)  CTRL-D  EOF  End of file  CTRL-M  CR  Carriage return (unchangeable)  CTRL-J  NL  Linefeed (unchangeable)  Figure 5-36. Characters that are handled specially in canonical mode.  The ERASE character allows the user to rub out the character just typed, it is usually the  backspace (CTRL-H). It is not added to the character queue but instead removes the  previous character from the queue. It should be echoed as a sequence of three characters,  backspace, space, and backspace, in order to remove the previous character from the  screen. If the previous character was a tab, erasing  it depends on how it was processed  when it was typed. If it is immediately expanded into spaces, some extra information is  needed to determine how far to back up. If the tab itself is stored in the input queue, it can  be removed and the entire line just output again. In most systems, backspacing will only  erase characters on the current line. It will not erase a carriage return and back up into the  previous line.  When the user notices an error at the start of the line being typed in it is often convenient to  erase the entire line and start again. The KILL character erases the entire line. Most systems  make the erased line vanish from the screen, but a few echo it plus a carriage return and  linefeed because some users like to see the old line. Consequently, how to echo KILL is a  matter of taste. As with ERASE it is usually not possible to go further back than the current  line. When a block of characters is killed, it may or may not he worth the trouble for the  driver to return buffers to the pool, if one is used.  Sometimes the ERASE or KILL characters must be entered as ordinary data. The LNEXT  character serves as an escape character . In UNIX CTRL-V is the default. As an example,  older UNIX systems often used the @ sign for KILL , but the Internet mail system uses  addresses of the form linda@cs.washington.edu . Someone who feels more comfortable  with older conventions might redefine KILL as @, but then need to enter an @ sign literally  to address email. This can be done by typing CTRL-V @. The CTRL-V itself can be  entered literally by typing CTRL-V CTRL-V. After seeing a CTRL-V, the driver sets a flag  saying that the next character is exempt from special processing. The LNEXT character  itself is not entered in the character queue.  To allow users to stop a screen image from scrolling out of view, control codes are  provided to freeze the screen and restart it later. In UNIX these are STOP , (CTRL-S) and  START , (CTRL-Q), respectively. They are not stored but are used to set and clear a flag in  the terminal data structure. Whenever output is attempted, the flag is inspected. If it is set,  no output occurs. Usually, echoing is also suppressed along with program output.  It is often necessary to kill a runaway program being debugged. The INTR (DEL) and QUIT  (CTRL-\) characters can be used for this purpose. In UNIX, DEL sends the SIGINT signal  to all the processes started up from the terminal. Implementing DEL can be quite tricky.  The hard part is getting the information from the driver to the part of the system that  handles signals, which, after all, has not asked for this information. CTRL-\ is similar to  DEL, except that it sends the SIGQUIT signal, which forces a core dump if not caught or  ignored. When either of these keys is struck, the driver should echo a carriage return and  linefeed and discard all accumulated input to allow for a fresh start. The default value for  INTR is often CTRL-C instead of DEL, since many programs use DEL interchangeably  with the backspace for editing.  Another special character is EOF (CTRL-D), which in UNIX causes any pending read  requests for the terminal to be satisfied with whatever is available in the buffer, even if the  buffer is empty. Typing CTRL-D at the start of a line causes the program to get a read of 0  bytes, which is conventionally interpreted as end-of-file and causes most programs to act  the same way as they would upon seeing end-of-file on an input file.  Some terminal drivers allow much fancier intraline editing than we have sketched here.  They have special control characters to erase a word, skip backward or forward characters  or words, go to the beginning or end of the line being typed, inserting text in the middle of  the line, and so forth. Adding all these functions to the terminal driver makes it much larger  and, furthermore, is wasted when using fancy screen editors that work in raw mode  anyway.  5.6.3 Output Software  Output is simpler than input. For the most part, the computer sends characters to the  terminal and they are displayed there. Usually, a block of characters, for example, a line, is  written to the terminal in one system call. The method that is commonly used for RS-232  terminals is to have output buffers associated with each terminal. The buffers can come  from the same pool as the input buffers, or be dedicated, as with input. When a program  writes to the terminal, the output is first copied to the buffer. Similarly, output from  echoing is also copied to the buffer. After all the output has been copied to the buffer, the  first character is output, and the driver goes to sleep. When the interrupt comes in, the next  character is output, and so on.  Screen editors and many other sophisticated programs need to be able to update the screen  in complex ways such as replacing one line in the middle of the screen. To accommodate  this need, most terminals support a series of commands to move the cursor, insert and  delete characters or lines at the cursor, etc. These commands are often called escape  sequences . In the heyday of the RS-232 terminal, there were hundreds of terminal types,  each with its own escape sequences. As a consequence, it was difficult to write software  that worked on more than one terminal type.  One solution, which was introduced in Berkeley UNIX, was a terminal database called  termcap . This software package defined a number of basic actions, such as moving the  cursor to (row , column ). To move the cursor to a particular location, the software, say, an  editor, used a generic escape sequence which was then converted to the actual escape  sequence for the terminal being written to. In this way, the editor worked on any terminal  that had an entry in the termcap database.  Eventually, the industry saw the need for standardization of the escape sequence, so an  ANSI standard was developed. A few of the values are shown in Fig. 5-37.  Consider how these escape sequences might be used by a text editor. Suppose that the user  types a command telling the editor to delete all of line 3 and then close up the gap between  lines 2 and 4. The editor might send the following escape sequence over the serial line to  the terminal:  ESC [ 3 ; 1 H ESC [ 0 K ESC [ 1 M  (where the spaces are used above only to separate the symbols; they are not transmitted).  This sequence moves the cursor to the start of line 3, erases the entire line, and then deletes  the now-empty line, causing all the lines starting at 5 to move up 1 line. Then what was line  4 becomes line 3; what was line 5 becomes line 4, and so on. Analogous escape sequences  can be used to add text to the middle of the display. Words and be added or removed in a  similar way.  Escape sequence  Meaning  ESC [ n A  Move up n lines  ESC [ n B  Move down n lines  SSC [ n C  Move right n spaces  ESC [ n D  Move left n spaces  ESC [ m ; n H  Move cursor to (m ,n )  ESC [ s J  Clear screen from cursor (0 to end, 1 from start, 2 all)  ESC [ s K  Clear line from cursor (0 to end, 1 from start, 2 all)  ESC [ n L  Insert n lines at cursor  ESC [ n M  Delete n lines at cursor  ESC [ n P  Delete n chars at cursor  ESC [ n @  Insert n chars at cursor  ESC [ n m  Enable rendition n (0=normal, 4=bold, 5=blinking, 7=reverse)  ESC M  Scroll the screen backward if the cursor is on the top line  Figure 5-37. The ANSI escape sequences accepted by the terminal driver on output. ESC  denotes the A | 5.6 ĐẦU CỬA ĐỊNH HƯỚNG KÝ TỰ  Mỗi máy tính đa năng đều có ít nhất một bàn phím và một màn hình (màn hình hoặc  màn hình) được sử dụng để giao tiếp với nó. Mặc dù bàn phím và màn hình trên một cá nhân  máy tính là những thiết bị riêng biệt về mặt kỹ thuật, chúng hoạt động chặt chẽ với nhau. Trên máy tính lớn,  thường xuyên có nhiều người dùng từ xa, mỗi người có một thiết bị chứa bàn phím và  màn hình kèm theo. Những thiết bị này trong lịch sử được gọi là thiết bị đầu cuối. chúng tôi sẽ tiếp tục  sử dụng thuật ngữ đó, ngay cả khi thảo luận về máy tính cá nhân (chủ yếu là do thiếu một thuật ngữ tốt hơn  thuật ngữ).  Thiết bị đầu cuối có nhiều dạng. Ba trong số các loại thường gặp nhất trong thực tế  ngày nay là  1. Thiết bị đầu cuối độc lập với giao diện nối tiếp RS-232 để sử dụng trên máy tính lớn.  2. Máy tính cá nhân hiển thị với giao diện người dùng đồ họa.  3. Thiết bị đầu cuối mạng.  Mỗi loại thiết bị đầu cuối có hốc sinh thái riêng của mình. Trong các phần sau chúng ta sẽ  mô tả lần lượt từng loại này.  5.6.1 Phần cứng đầu cuối RS-232  Thiết bị đầu cuối RS-232 là thiết bị phần cứng chứa cả bàn phím và màn hình và  giao tiếp bằng giao diện nối tiếp, từng bit một (xem Hình 5-34). Các thiết bị đầu cuối này sử dụng  đầu nối 9 chân hoặc 25 chân, trong đó một chân được sử dụng để truyền dữ liệu, một chân dành cho  nhận dữ liệu và một chân được nối đất. Các chân khác dành cho các chức năng điều khiển khác nhau, hầu hết  trong số đó không được sử dụng. Các dòng trong đó các ký tự được gửi từng bit một (trái ngược với 8  các bit song song theo cách máy in được giao tiếp với PC) được gọi là đường nối tiếp . Tất cả modem  cũng sử dụng giao diện này. Trên UNIX, các dòng nối tiếp có tên như /dev/tty1 và /dev/tty2 . TRÊN  Windows chúng có các tên như COM1 và COM2.  Để gửi một ký tự qua đường nối tiếp đến thiết bị đầu cuối RS-232 hoặc modem, máy tính phải  truyền 1 bit tại một thời điểm, bắt đầu bằng bit bắt đầu và theo sau là 1 hoặc 2 bit dừng để phân định  nhân vật. Một bit chẵn lẻ cung cấp khả năng phát hiện lỗi cơ bản cũng có thể được chèn vào  trước các bit dừng, mặc dù điều này thường chỉ được yêu cầu để liên lạc với  các hệ thống máy tính lớn.  Hình 5-34. Thiết bị đầu cuối RS-232 giao tiếp với máy tính qua giao tiếp  dòng, từng chút một.  Đầu cuối RS-232 vẫn thường được sử dụng trong thế giới máy tính lớn để cho phép người dùng từ xa  giao tiếp với máy tính lớn, đôi khi sử dụng modem và đường dây điện thoại. Họ  được tìm thấy trong ngành hàng không, ngân hàng và các ngành công nghiệp khác. Ngay cả khi chúng được thay thế bằng  máy tính cá nhân, PC thường chỉ mô phỏng các thiết bị đầu cuối RS-232 cũ để tránh  phải thay đổi phần mềm máy tính lớn.  Những thiết bị đầu cuối này cũng từng thống trị thế giới máy tính mini. Rất nhiều phần mềm dành cho  các hệ thống phát triển trong thời kỳ này đều dựa trên các thiết bị đầu cuối này. Ví dụ, tất cả UNIX  hệ thống hỗ trợ loại thiết bị này.  Tuy nhiên, thậm chí còn quan trọng hơn, nhiều hệ thống UNIX hiện tại (và các hệ thống khác) cung cấp  tùy chọn tạo một cửa sổ bao gồm một số dòng văn bản. Nhiều  các lập trình viên hầu như chỉ làm việc ở chế độ văn bản trong các cửa sổ như vậy, ngay cả trên trang cá nhân  máy tính hoặc máy trạm cao cấp. Các cửa sổ này thường mô phỏng một số RS-232  thiết bị đầu cuối (hoặc tiêu chuẩn ANSI cho loại thiết bị đầu cuối này) để chúng có thể chạy phần lớn hiện có  cơ sở phần mềm đã được viết cho các thiết bị đầu cuối như vậy. Trong suốt nhiều năm, phần mềm này,  chẳng hạn như trình chỉnh sửa vi và emacs, đã được gỡ lỗi hoàn toàn và cực kỳ ổn định,  các lập trình viên thuộc tính đánh giá cao.  Phần mềm bàn phím và thiết bị đầu cuối cho các cửa sổ mô phỏng thiết bị đầu cuối này giống như  cho các thiết bị đầu cuối thực sự. Vì các trình giả lập thiết bị đầu cuối này đang được sử dụng rộng rãi nên phần mềm  vẫn quan trọng, vì vậy chúng tôi sẽ mô tả nó trong hai phần sau.  Thiết bị đầu cuối RS-232 được định hướng theo ký tự. Điều này có nghĩa là màn hình hoặc cửa sổ  hiển thị một số dòng văn bản nhất định, mỗi dòng có kích thước tối đa. Một kích thước điển hình là 25 dòng  gồm 80 ký tự mỗi ký tự. Mặc dù một vài ký tự đặc biệt đôi khi được hỗ trợ, những ký tự này  thiết bị đầu cuối (và trình giả lập) về cơ bản chỉ là văn bản.  Vì cả máy tính và thiết bị đầu cuối đều hoạt động bên trong với toàn bộ ký tự nhưng phải  giao tiếp qua đường nối tiếp từng chút một, chip đã được phát triển để thực hiện  chuyển đổi từ ký tự sang sê-ri và từ sê-ri sang ký tự. Chúng được gọi là UART (Universal  Bộ thu phát không đồng bộ). UART được gắn vào máy tính bằng cách cắm  Thẻ giao diện RS-232 vào bus như minh họa trong Hình 5-34. Trên nhiều máy tính, một hoặc  hai cổng nối tiếp được tích hợp vào bo mạch chủ.  Để hiển thị một ký tự, trình điều khiển đầu cuối ghi ký tự đó vào thẻ giao diện, trong đó  nó được lưu vào bộ đệm và sau đó được chuyển ra khỏi dòng nối tiếp từng bit một bởi UART. Vì  ví dụ, đối với một modem analog hoạt động ở tốc độ 56.000 bps, chỉ mất hơn 179 giây để gửi một  tính cách. Do tốc độ truyền chậm này, trình điều khiển thường xuất ra một ký tự  đến thẻ RS-232 và chặn, chờ ngắt do giao diện tạo ra khi  ký tự đã được truyền và UART có thể chấp nhận ký tự khác  tay đua. Các  UART có thể gửi và nhận các ký tự đồng thời. Một ngắt cũng được tạo ra khi  một ký tự được nhận và thông thường một số lượng nhỏ ký tự đầu vào có thể được lưu vào bộ đệm. Các  trình điều khiển đầu cuối phải kiểm tra một thanh ghi khi nhận được một ngắt để xác định nguyên nhân  của ngắt. Một số thẻ giao diện có CPU và bộ nhớ và có thể xử lý nhiều  dòng, chiếm phần lớn tải I/O từ CPU chính.  Thiết bị đầu cuối RS-232 có thể được chia thành ba loại. Những cái đơn giản nhất là bản cứng  (tức là, in ấn) thiết bị đầu cuối. Các ký tự gõ trên bàn phím được truyền đến máy tính.  Các ký tự do máy tính gửi được in trên giấy. Các thiết bị đầu cuối này đã lỗi thời và  hiếm khi được nhìn thấy nữa ngoại trừ máy in cấp thấp.  Các thiết bị đầu cuối CRT câm hoạt động theo cách tương tự, chỉ với một màn hình thay vì giấy. đó là  thường được gọi là “tty thủy tinh” vì chúng có chức năng giống như tty bản cứng. (thuật ngữ  "tty" là tên viết tắt của Teletype, một công ty trước đây đi tiên phong trong lĩnh vực máy tính  kinh doanh thiết bị đầu cuối: “tty” có nghĩa là bất kỳ thiết bị đầu cuối nào.) ttys bằng thủy tinh cũng đã lỗi thời.  Các thiết bị đầu cuối CRT thông minh trên thực tế là các máy tính chuyên dụng thu nhỏ. Họ có một CPU  và bộ nhớ và chứa phần mềm, thường là trong ROM. Từ hệ điều hành  quan điểm, sự khác biệt chính giữa tty thủy tinh và thiết bị đầu cuối thông minh là  sau này hiểu các trình tự thoát nhất định. Ví dụ: bằng cách gửi ASCII ESC  ký tự (0x1B), theo sau là nhiều ký tự khác, có thể di chuyển  con trỏ đến bất kỳ vị trí nào trên màn hình, chèn văn bản vào giữa màn hình, v.v.  Thiết bị đầu cuối thông minh là thiết bị được sử dụng trong các hệ thống máy tính lớn và là thiết bị được mô phỏng bởi  các hệ điều hành khác. Đó là phần mềm của họ mà chúng ta sẽ thảo luận dưới đây.  5.6.2 Phần mềm nhập liệu  Bàn phím và màn hình là những thiết bị gần như độc lập, vì vậy chúng tôi sẽ xử lý chúng riêng biệt  đây. Tuy nhiên, chúng không hoàn toàn độc lập vì các ký tự được gõ nói chung là  hiển thị trên màn hình.  Công việc cơ bản của trình điều khiển bàn phím là thu thập đầu vào từ bàn phím và chuyển cho người dùng  các chương trình khi chúng đọc từ thiết bị đầu cuối. Hai triết lý có thể có thể được áp dụng cho  người lái xe. Trong phần đầu tiên, công việc của trình điều khiển chỉ là chấp nhận đầu vào và chuyển nó lên trên  không sửa đổi. Một chương trình đọc từ thiết bị đầu cuối sẽ nhận được một chuỗi mã ASCII thô.  (Cung cấp cho chương trình người dùng các số khóa là quá thô sơ, cũng như mang tính máy móc cao  sự phụ thuộc.)  Triết lý này rất phù hợp với nhu cầu của các trình chỉnh sửa màn hình phức tạp như emacs,  cho phép người dùng liên kết một hành động tùy ý với bất kỳ ký tự hoặc chuỗi ký tự nào.  Tuy nhiên, điều đó có nghĩa là nếu người dùng nhập dste thay vì ngày và sau đó sửa lỗi  bằng cách gõ ba khoảng lùi và ate , theo sau là dấu xuống dòng, chương trình người dùng sẽ được  đưa ra tất cả 11 mã ASCII đã nhập, như sau:  d s t e a t e CR  Không phải tất cả các chương trình muốn nhiều chi tiết này. Thường thì họ chỉ muốn đầu vào được sửa chứ không phải  trình tự chính xác về cách nó được sản xuất. Quan sát này dẫn đến triết lý thứ hai:  trình điều khiển xử lý tất cả các chỉnh sửa nội tuyến và chỉ cung cấp các dòng đã sửa cho người dùng  các chương trình. Triết lý đầu tiên là định hướng nhân vật; cái thứ hai là định hướng theo dòng.  Ban đầu chúng được gọi tương ứng là chế độ sống và chế độ nấu chín. POSIX  tiêu chuẩn sử dụng thuật ngữ chế độ chính tắc ít đẹp mắt hơn để mô tả chế độ định hướng theo đường.  Chế độ không chính tắc tương đương với chế độ thô, mặc dù nhiều chi tiết về hoạt động của thiết bị đầu cuối  có thể thay đổi. Các hệ thống tương thích với POSIX cung cấp một số chức năng thư viện hỗ trợ  chọn một trong hai chế độ và thay đổi nhiều khía cạnh của cấu hình thiết bị đầu cuối.  Nhiệm vụ đầu tiên của trình điều khiển bàn phím là thu thập các ký tự. Nếu mỗi lần nhấn phím gây ra một  ngắt, trình điều khiển có thể thu được ký tự trong khi ngắt. Nếu các ngắt được bật  vào tin nhắn bằng phần mềm cấp thấp, có thể đưa ký tự mới thu được  trong tin nhắn. Ngoài ra, nó có thể được đặt trong một bộ đệm nhỏ trong bộ nhớ và thông báo  được sử dụng để nói với người lái xe rằng một cái gì đó đã đến. Cách tiếp cận thứ hai thực sự an toàn hơn nếu một  tin nhắn chỉ có thể được gửi đến một tiến trình đang đợi và có một số khả năng là bàn phím  trình điều khiển có thể vẫn đang bận với ký tự trước đó.  Nếu thiết bị đầu cuối ở chế độ chính tắc (đã nấu chín), các ký tự phải được lưu trữ cho đến khi toàn bộ dòng  đã được tích lũy, bởi vì người dùng sau đó có thể quyết định xóa một phần của nó. Thậm chí nếu  thiết bị đầu cuối ở chế độ thô, chương trình có thể chưa yêu cầu đầu vào, vì vậy  các ký tự phải được đệm để cho phép nhập trước. (Người thiết kế hệ thống không cho phép người dùng  để gõ xa về phía trước phải bị hắc ín và có lông, hoặc tệ hơn nữa, buộc phải sử dụng chính họ  hệ thống.)  Hai cách tiếp cận để đệm ký tự là phổ biến. Trong cái đầu tiên, trình điều khiển chứa một  nhóm bộ đệm trung tâm, mỗi bộ đệm có thể chứa 10 ký tự. Liên kết với mỗi  thiết bị đầu cuối là một cấu trúc dữ liệu, trong đó có chứa, trong số các mục khác, một điểm  er đến chuỗi  bộ đệm cho đầu vào được thu thập từ thiết bị đầu cuối đó. Khi nhiều ký tự được nhập, nhiều bộ đệm hơn  được mua và treo trên chuỗi. Khi các ký tự được chuyển đến một chương trình người dùng,  bộ đệm được loại bỏ và đưa trở lại nhóm trung tâm.  Cách tiếp cận khác là thực hiện đệm trực tiếp trong chính cấu trúc dữ liệu đầu cuối, với  không có vùng đệm trung tâm. Vì thông thường người dùng gõ một lệnh sẽ mất một  một chút (ví dụ, biên dịch lại và liên kết một chương trình nhị phân lớn) và sau đó nhập một vài dòng  phía trước, để đảm bảo an toàn, người lái xe nên phân bổ khoảng 200 ký tự cho mỗi thiết bị đầu cuối. trong một  hệ thống chia sẻ thời gian quy mô lớn với 100 thiết bị đầu cuối, phân bổ 20K mọi lúc cho loại  phía trước rõ ràng là quá mức cần thiết, do đó, một nhóm bộ đệm trung tâm có không gian cho 5K có lẽ là  đủ. Mặt khác, bộ đệm dành riêng cho mỗi thiết bị đầu cuối giúp trình điều khiển đơn giản hơn (không  quản lý danh sách được liên kết) và được ưu tiên trên các máy tính cá nhân chỉ có một  bàn phím. Hình 5-35 cho thấy sự khác biệt giữa hai phương pháp này.  Mặc dù bàn phím và màn hình là những thiết bị tách biệt hợp lý nhưng nhiều người dùng đã phát triển  quen với việc nhìn thấy các ký tự mình vừa gõ xuất hiện trên màn hình. Một số (lớn tuổi)  thiết bị đầu cuối bắt buộc bằng cách tự động hiển thị (trong phần cứng) bất cứ thứ gì vừa được nhập,  điều này không chỉ gây phiền toái khi nhập mật khẩu mà còn hạn chế rất nhiều  tính linh hoạt của các trình soạn thảo phức tạp và các chương trình khác. May mắn thay, với hầu hết các thiết bị đầu cuối,  không có gì tự động hiển thị khi nhấn phím. Nó hoàn toàn phụ thuộc vào phần mềm trong  máy tính để hiển thị ký tự, nếu muốn. Quá trình này được gọi là tiếng vang.  Hình 5-35. ( a ) Vùng đệm trung tâm. (b) Bộ đệm chuyên dụng cho mỗi thiết bị đầu cuối.  Việc lặp lại phức tạp bởi thực tế là một chương trình có thể đang ghi lên màn hình trong khi  người dùng đang gõ. Ít nhất, trình điều khiển bàn phím phải tìm ra nơi để đặt cái mới  đầu vào mà không bị ghi đè bởi đầu ra của chương trình.  Tiếng vang cũng trở nên phức tạp khi phải hiển thị hơn 80 ký tự trên một  màn hình với các dòng 80 ký tự (hoặc một số khác). Tùy thuộc vào ứng dụng,  quấn quanh dòng tiếp theo có thể phù hợp. Một số trình điều khiển chỉ cắt ngắn dòng đến 80  các ký tự bằng cách loại bỏ tất cả các ký tự ngoài cột 80.  Một vấn đề khác là xử lý tab. Trình điều khiển thường tính toán vị trí của con trỏ  hiện tại, có tính đến cả đầu ra từ các chương trình và đầu ra từ  lặp lại và tính toán số khoảng trống thích hợp sẽ được lặp lại.  Bây giờ chúng ta đến với vấn đề tương đương thiết bị. Theo logic, ở cuối một dòng văn bản,  một người muốn xuống dòng, để di chuyển con trỏ về cột 1 và một nguồn cấp dữ liệu, để  chuyển sang dòng tiếp theo. Yêu cầu người dùng nhập cả hai ở cuối mỗi dòng sẽ không bán được  tốt (mặc dù một số thiết bị đầu cuối có khóa tạo ra cả hai, với 50% cơ hội  làm như vậy theo thứ tự mà phần mềm muốn chúng). Tùy thuộc vào trình điều khiển để chuyển đổi  bất cứ thứ gì có định dạng nội bộ tiêu chuẩn được sử dụng bởi hệ điều hành.  Nếu biểu mẫu tiêu chuẩn chỉ để lưu trữ nguồn cấp dữ liệu (quy ước UNIX), thì dấu xuống dòng sẽ trả về  nên được biến thành nguồn cấp dữ liệu. Nếu định dạng nội bộ là để lưu trữ cả hai (Windows  quy ước), thì trình điều khiển sẽ tạo nguồn cấp dữ liệu khi nó nhận được dấu xuống dòng và  vận chuyển trở lại khi nó nhận được một nguồn cấp dữ liệu. Bất kể quy ước nội bộ là gì, thiết bị đầu cuối  có thể yêu cầu lặp lại cả nguồn cấp dữ liệu và xuống dòng để có được màn hình  được cập nhật đúng cách. Vì một máy tính lớn có thể có nhiều loại khác nhau  thiết bị đầu cuối được kết nối với nó, tùy thuộc vào trình điều khiển bàn phím để có được tất cả các vận chuyển khác nhau  kết hợp trở lại/dòng nguồn được chuyển đổi thành tiêu chuẩn hệ thống nội bộ và sắp xếp cho tất cả  lặp lại để được thực hiện đúng. Khi hoạt động ở chế độ chính tắc, một số ký tự đầu vào  có ý nghĩa đặc biệt. Hình 5-36 hiển thị tất cả các ký tự đặc biệt theo yêu cầu của POSIX.  Giá trị mặc định là tất cả các ký tự điều khiển không xung đột với kiểu nhập văn bản hoặc mã được sử dụng  bởi các chương trình, nhưng tất cả ngoại trừ hai cái cuối cùng có thể được thay đổi dưới sự kiểm soát của chương trình.  Tính cách  tên POSIX  Bình luận  CTRL-H  TẨY XÓA  Backspace một ký tự  CTRL-U  GIẾT  Xóa toàn bộ dòng đang gõ  CTRL-V  TIẾP THEO  Diễn giải ký tự tiếp theo theo nghĩa đen  CTRL-S  DỪNG LẠI  Dừng đầu ra  CTRL-Q  BẮT ĐẦU  bắt đầu xuất  DEL  đầu vào  Quá trình ngắt (SIGINT)  ĐIỀU KHIỂN-\  TỪ BỎ  Buộc kết xuất lõi (SIGQUIT)  CTRL-D  EOF  Phần cuối của tập tin  CTRL-M  CR  Vận chuyển trở lại (không thể thay đổi)  CTRL-J  NL  Nguồn cấp dữ liệu (không thể thay đổi)  Hình 5-36. Các ký tự được xử lý đặc biệt trong chế độ chuẩn.  Ký tự ERASE cho phép người dùng xóa ký tự vừa gõ, nó thường là ký tự  xóa lùi (CTRL-H). Nó không được thêm vào hàng ký tự mà thay vào đó loại bỏ  ký tự trước đó từ hàng đợi. Nó sẽ được lặp lại dưới dạng một chuỗi gồm ba ký tự,  xóa lùi, dấu cách và xóa lùi, để xóa ký tự trước đó khỏi  màn hình. Nếu ký tự trước đó là một tab, xóa  nó phụ thuộc vào cách nó được xử lý  khi nó được gõ. Nếu nó ngay lập tức được mở rộng vào không gian, một số thông tin bổ sung là  cần thiết để xác định khoảng cách để sao lưu. Nếu bản thân tab được lưu trữ trong hàng đợi đầu vào, nó có thể  bị xóa và toàn bộ dòng chỉ xuất lại. Trong hầu hết các hệ thống, khoảng lùi sẽ chỉ  xóa các ký tự trên dòng hiện tại. Nó sẽ không xóa dấu xuống dòng và sao lưu vào  dòng trước.  Khi người dùng nhận thấy lỗi ở đầu dòng được nhập, điều đó thường thuận tiện để  xóa toàn bộ dòng và bắt đầu lại. Ký tự KILL xóa toàn bộ dòng. Hầu hết các hệ thống  làm cho dòng bị xóa biến mất khỏi màn hình, nhưng một số tiếng vang cộng với dấu xuống dòng và  nguồn cấp dữ liệu vì một số người dùng muốn xem dòng cũ. Do đó, làm thế nào để lặp lại KILL là một  vấn đề của hương vị. Như với ERASE, thường không thể quay lại xa hơn hiện tại  đường kẻ. Khi một khối nhân vật bị giết, anh ta có thể đáng hoặc không đáng để gây rắc rối cho  trình điều khiển để trả lại bộ đệm cho nhóm, nếu bộ đệm được sử dụng.  Đôi khi các ký tự ERASE hoặc KILL phải được nhập dưới dạng dữ liệu thông thường. Lnext  ký tự đóng vai trò là ký tự thoát. Trong UNIX CTRL-V là mặc định. Như một ví dụ,  các hệ thống UNIX cũ hơn thường sử dụng ký hiệu @ cho KILL , nhưng hệ thống thư Internet sử dụng  địa chỉ có dạng linda@cs.washington.edu . Ai đó cảm thấy thoải mái hơn  với các quy ước cũ hơn có thể xác định lại KILL là @, nhưng sau đó cần nhập ký hiệu @ theo nghĩa đen  để giải quyết email. Điều này có thể được thực hiện bằng cách gõ CTRL-V @. Bản thân CTRL-V có thể là  được nhập theo nghĩa đen bằng cách gõ CTRL-V CTRL-V. Sau khi nhìn thấy CTRL-V, trình điều khiển sẽ đặt cờ  nói rằng ký tự tiếp theo được miễn xử lý đặc biệt. Ký tự Lnext  chính nó không được nhập vào hàng ký tự.  Để cho phép người dùng ngăn hình ảnh màn hình cuộn ra khỏi chế độ xem, các mã kiểm soát là  được cung cấp để đóng băng màn hình và khởi động lại sau. Trong UNIX, đây là STOP , (CTRL-S) và  BẮT ĐẦU , (CTRL-Q), tương ứng. Chúng không được lưu trữ nhưng được sử dụng để đặt và xóa cờ trong  cấu trúc dữ liệu đầu cuối. Bất cứ khi nào đầu ra được thử, cờ sẽ được kiểm tra. Nếu nó được thiết lập,  không có đầu ra xảy ra. Thông thường, tiếng vang cũng bị triệt tiêu cùng với đầu ra của chương trình.  Nó thường là cần thiết để giết một chương trình chạy trốn đang được gỡ lỗi. INTR (DEL) và QUIT  (CTRL-\) ký tự có thể được sử dụng cho mục đích này. Trong UNIX, DEL gửi tín hiệu SIGINT  cho tất cả các quy trình được khởi động từ thiết bị đầu cuối. Việc triển khai DEL có thể khá phức tạp.  Phần khó là lấy thông tin từ trình điều khiển đến một phần của hệ thống  xử lý các tín hiệu, mà sau tất cả, đã không yêu cầu thông tin này. CTRL-\ tương tự như  DEL, ngoại trừ việc nó gửi tín hiệu SIGQUIT, tín hiệu này buộc kết xuất lõi nếu không bị bắt hoặc  làm ngơ. Khi một trong hai phím này được nhấn, người lái xe sẽ lặp lại tiếng xuống dòng và  nguồn cấp dữ liệu và loại bỏ tất cả đầu vào tích lũy để cho phép bắt đầu mới. Giá trị mặc định cho  INTR thường là CTRL-C thay vì DEL, vì nhiều chương trình sử dụng DEL thay thế cho nhau  với khoảng lùi để chỉnh sửa.  Một ký tự đặc biệt khác là EOF (CTRL-D), ký tự này trong UNIX gây ra bất kỳ thao tác đọc đang chờ xử lý nào  yêu cầu thiết bị đầu cuối hài lòng với bất kỳ thứ gì có sẵn trong bộ đệm, ngay cả khi  bộ đệm trống. Nhập CTRL-D ở đầu dòng khiến chương trình đọc 0  byte, thường được hiểu là phần cuối của tệp và khiến hầu hết các chương trình hoạt động  giống như cách họ làm khi nhìn thấy phần cuối của tệp trên tệp đầu vào.  Một số trình điều khiển đầu cuối cho phép chỉnh sửa nội tuyến đẹp hơn nhiều so với những gì chúng tôi đã phác thảo ở đây.  Họ có các ký tự điều khiển đặc biệt để xóa một từ, bỏ qua các ký tự lùi hoặc tiến  hoặc từ, chuyển đến đầu hoặc cuối dòng đang nhập, chèn văn bản vào giữa  dòng, v.v. Thêm tất cả các chức năng này vào trình điều khiển đầu cuối làm cho nó lớn hơn nhiều  và hơn nữa, bị lãng phí khi sử dụng các trình chỉnh sửa màn hình ưa thích hoạt động ở chế độ thô  Dẫu sao thì.  5.6.3 Phần mềm đầu ra  Đầu ra đơn giản hơn đầu vào. Phần lớn, máy tính gửi các ký tự đến  terminal và chúng được hiển thị ở đó. Thông thường, một khối ký tự, ví dụ, một dòng, là  được ghi vào thiết bị đầu cuối trong một cuộc gọi hệ thống. Phương pháp thường được sử dụng cho RS-232  thiết bị đầu cuối là có bộ đệm đầu ra được liên kết với mỗi thiết bị đầu cuối. Bộ đệm có thể đến  từ cùng một nhóm với bộ đệm đầu vào hoặc được dành riêng như với đầu vào. Khi một chương trình  ghi vào thiết bị đầu cuối, đầu ra đầu tiên được sao chép vào bộ đệm. Tương tự, đầu ra từ  tiếng vang cũng được sao chép vào bộ đệm. Sau khi tất cả đầu ra đã được sao chép vào bộ đệm,  ký tự đầu tiên được xuất ra và trình điều khiển chuyển sang chế độ ngủ. Khi ngắt xuất hiện, tiếp theo  ký tự là đầu ra, v.v.  Trình chỉnh sửa màn hình và nhiều chương trình phức tạp khác cần có khả năng cập nhật màn hình  theo những cách phức tạp như thay thế một dòng ở giữa màn hình. để chứa  nhu cầu này, hầu hết các thiết bị đầu cuối đều hỗ trợ một loạt lệnh để di chuyển con trỏ, chèn và  xóa các ký tự hoặc dòng tại con trỏ,  v.v. Các lệnh này thường được gọi là thoát  trình tự . Vào thời hoàng kim của thiết bị đầu cuối RS-232, có hàng trăm loại thiết bị đầu cuối,  mỗi cái có trình tự thoát riêng. Kết quả là, rất khó để viết phần mềm  đã hoạt động trên nhiều loại thiết bị đầu cuối.  Một giải pháp đã được giới thiệu trong Berkeley UNIX, là một cơ sở dữ liệu đầu cuối được gọi là  thuật ngữ . Gói phần mềm này đã xác định một số hành động cơ bản, chẳng hạn như di chuyển  con trỏ tới (hàng, cột). Để di chuyển con trỏ đến một vị trí cụ thể, phần mềm, giả sử, một  trình chỉnh sửa, đã sử dụng trình tự thoát chung, sau đó được chuyển đổi thành trình tự thoát thực tế  trình tự cho thiết bị đầu cuối được ghi vào. Bằng cách này, trình chỉnh sửa đã hoạt động trên bất kỳ thiết bị đầu cuối nào  đã có một mục trong cơ sở dữ liệu termcap.  Cuối cùng, ngành công nghiệp nhận thấy sự cần thiết phải tiêu chuẩn hóa chuỗi thoát, vì vậy một  Tiêu chuẩn ANSI đã được phát triển. Một số giá trị được hiển thị trong Hình 5-37.  Xem xét cách trình soạn thảo văn bản có thể sử dụng các chuỗi thoát này. Giả sử rằng người dùng  gõ một lệnh yêu cầu trình chỉnh sửa xóa tất cả dòng 3 và sau đó thu hẹp khoảng cách giữa  dòng 2 và 4. Trình chỉnh sửa có thể gửi trình tự thoát sau qua dòng nối tiếp tới  thiết bị đầu cuối:  ESC [ 3 ; 1 H ESC [ 0 K ESC [ 1 M  (trong đó các khoảng trắng được sử dụng ở trên chỉ để phân tách các ký hiệu; chúng không được truyền đi).  Trình tự này di chuyển con trỏ đến đầu dòng 3, xóa toàn bộ dòng và sau đó xóa  dòng hiện trống, làm cho tất cả các dòng bắt đầu từ 5 di chuyển lên 1 dòng. Sau đó, dòng là gì  4 trở thành dòng 3; dòng 5 trở thành dòng 4, v.v. Trình tự thoát tương tự  có thể được sử dụng để thêm văn bản vào giữa màn hình. Các từ và được thêm hoặc bớt trong một  cách giống nhau.  Trình tự thoát  Nghĩa  ESC [ n A  Di chuyển lên n dòng  ESC [ n B  Xuống n dòng  SSC [ n C  Di chuyển sang phải n dấu cách  ESC [ nD  Di chuyển sang trái n dấu cách  ESC [ m ; n H  Di chuyển con trỏ đến (m ,n )  ESC [ s J  Xóa màn hình khỏi con trỏ (0 đến cuối, 1 từ đầu, 2 tất cả)  ESC [ s K  Xóa dòng khỏi con trỏ (0 đến cuối, 1 từ đầu, 2 tất cả)  ESC [ n L  Chèn n dòng vào con trỏ  ESC [ n M  Xóa n dòng tại con trỏ  ESC [ n P  Xóa n ký tự tại con trỏ  ESC [ n @  Chèn n ký tự vào con trỏ  ESC [ n m  Bật hiển thị n (0=bình thường, 4=đậm, 5=nhấp nháy, 7=đảo ngược)  ESC M  Cuộn ngược màn hình nếu con trỏ ở dòng trên cùng  Hình 5-37. Trình tự thoát ANSI được trình điều khiển đầu cuối chấp nhận ở đầu ra. THOÁT  ký hiệu là A |