**TP0 - Guide de débogage de xv6**

**Débogage : 调试**

**L’objectif de ce TP, à faire chez vous avant le premier TP sur l’ordonnancement, est de vous familiariser avec le noyau xv6, notamment avec les méthodes de débogage qui vous seront bien utiles.**

1. **Interblocage**

**Interblocage : 互锁，死结。当两个以上的运算单元，双方都在等待对方停止运行，以取得系统资源，但是没有一方提前退出时，这种情况被称为死结(Deadlock). 在多任务操作系统中，操作系统为了协调不同的进程，能否取得系统资源时，为了让系统原作，就必须要解决这个问题。**

**Bd :**

**Virtio disk init 0 :虚拟地址**

**Hart : HART通讯协定，HART Communication Protocol**

**Proc : Linux系统上的/proc目录是一种文件系统，即proc文件系统。 与其它常见的文件系统不同的是，/proc是一种伪文件系统（也即虚拟文件系统），存储的是当前内核运行状态的一系列特殊文件，用户可以通过这些文件查看有关系统硬件及当前正在运行进程的信息，甚至可以通过更改其中某些文件来改变内核的运行状态。**

**Verrou : 锁扣**

**Détenir：占有**

**Acquérir: 获得**

**Dump: 转储**

**Manifestation: 表明**

**Linux grep 命令用于查找文件里符合条件的字符串**

**Décommettre：松开绳子**

**La notion de panique signifie que le noyau a détecté une erreur, de laquelle il n’est pas capable de récupérer : le système est bloqué à partir de ce moment.**

**HINT: restart xv6 using “make qemu-gdb”, type “b panic” (to set breakpoint in panic) in the gdb window, followed by “c” (continue), and when the kernel hits the breakpoint, type “bt” to get a backtrace.**

1. **Organisation du code de xv6**

**— kernel/ : c’est là que se trouve le code du noyau xv6 à proprement parler : c’est le code qui s’exécutera en mode superviseur, et qui implémente les différents gestionnaires d’interruption, les appels systèmes, la gestion de la mémoire virtuelle, l’ordonnancement. . .**

**— user/ : c’est là que sont écrits les programmes de xv6 qui sont accessibles en mode utilisateur : le Shell (user/sh.c), le programme echo (user/echo.c), le programme ls (user/ls.c), ainsi que différents programmes de tests (user/usertests.c (inclus dans xv6), user/mutest.c, user/nice-exit.c (ajoutés par nos soins pour le TP1)).**

**— mkfs/ : ce répertoire contient du code qui ne fait pas partie du noyau xv6 à proprement parler. Il s’agit d’un utilitaire qui construit un système de fichiers virtuel, à partir du code du noyau compilé ainsi que de l’ensemble des programmes utilisateur présents dans le répertoire user. Vous n’aurez jamais à toucher au code de ce répertoire.**

**Succinctement : 简短的，简明扼要地**

**Exec.c: exec () system call**

**Proc.c(): processes and scheduling**

1. **Appels système**

**Survenir: 突如其来，突然发生**

**Trappe:**

**Basculer: 摇摆**

**on inspecte la valeur du registre scause pour déterminer le type de trappe que l’on**

**doit gérer. D’après le manuel RISC-V, et les slides montrées en cours, la cause numéro 8 correspond à un appel système.**

**La fonction syscall *dispatche* les différents appels système en utilisant le numéro de l’appel système, passé dans le registre a7 (par convention).**

1. **Outils de débogage à l’intérieur du noyau**

**Bien que moins élégante que l’utilisation d’un débogueur comme gdb, insérer des printf à certains endroits bien choisis du code reste une manière relativement efficace de comprendre la source de certains bugs.**

**Avec parcimonie : 精打细算地**

**Crouler : 陷入**

**Malin : 机灵的**

**— procdump() pour afficher la liste des processus courante et leur état (pensez à inclure defs.h). Sans modifier le code du noyau, vous pouvez également afficher cette table en tapant Ctrl +P dans la console de xv6.— priodump() pour afficher la liste des files de priorité de l’ordonnanceur que vous développerez au TP 1 (pensez à inclure defs.h). Le raccourci Ctrl + Q permet d’afficher cette table.**

**— dump\_locks() pour afficher la liste des verrous détenus à un instant donné (pensez à inclure spinlock.h). Le raccourci “Ctrl + L” affiche cette table.**

**Notez également que la version de xv6 que nous vous proposons expose 3 consoles virtuelles. Vous pouvez passer d’une console virtuelle à la suivante en tapant Ctrl + S.**

1. **Déboguer avec gdb**

**Embêter: 烦扰**

**Soigneusement：细心地**

**Il s’agit ici de 0xf, c’est-à-dire store/AMO page fault, une faute de page en écriture. En d’autres termes, on a essayé d’écrire à une adresse qui n’est pas valide.**

**Fautif : 有错误的**

**La dernière commande layout split vous donne une vue du code en cours d’exécution avec trois fenêtres : la première affiche les lignes de code C actuellement exécutées, la seconde affiche les instructions assembleur actuellement exécutées, et la troisième est l’invite de commandes (là où vous pouvez taper... des commandes !).**

**À partir de là, vous pouvez :**

**— avancer d’une instruction assembleur en tapant si (step instruction)**

**— avancer d’une instruction assembleur, mais « sauter au-dessus des appels de fonctions », i.e. faire une série de pas jusqu’à arriver à l’instruction se situant juste après une instruction d’appel de fonction, en tapant ni (next instruction)**

**— avancer d’une instruction C en tapant s (step)**

**— avancer d’une instruction C, mais « sauter au-dessus des appels de fonctions », i.e. faire une série de pas jusqu’à arriver à l’instruction se situant juste après une instruction d’appel de fonction, en tapant n (next)Hmm : “used to express reflection, uncertainty, or hesitation”**

**Vous pouvez afficher la valeur d’une variable C de nom var en tapant la commande p var. On peut également afficher la valeur d’un registre, en préfixant son nom par $.**

**Vous pouvez examiner une région de la mémoire avec la commande x .**

**La commande x/24x $sp examine 24 morceaux de données, chacune de la taille d’un mot (4 octets), affichés en hexadécimal, à partir de l’adresse de départ $sp .**

**L’ensemble des registres peut être visualisé (si vous avez de bons yeux et/ou un grand écran) en tapant layout regs.**