Formation debugging, profiling, tracing et analyse de performance sous Linux

Durée de la formation

3 jours – 24 h

Langue

Transparents	Anglais
Présentation	Français
	Anglais
	Italien

Formateur

Un des ingénieurs suivants

- Alexis Lothoré
- Luca Ceresoli
- Olivier Benjamin

Contact

training@bootlin.com

+33 4 84 25 80 96

bootlin

bootlin.com

Formation en présentiel

Public visé

Sociétés et ingénieurs intéressés dans le debug, profiling et tracing de systèmes et d'applications Linux, afin d'analyser et résoudre des problèmes de performance ou de latence.

Objectifs opérationnels

- Être capable de comprendre les principaux concepts de Linux qui sont liés à l'analyse de performance : processus, threads, gestion de la mémoire, mémoire virtuelle, contextes d'exécution, etc.
- Être capable d'analyser pourquoi un système est chargé et quels sont les éléments qui contribuent à cette charge avec les outils usuels d'observabilité sous Linux.
- Être capable de débugger une application espace utilisateur avec *gdb*, soit en direct soit *post-mortem* suite à un crash, et analyser le contenu de binaires ELF.
- Être capable d'utiliser le *tracing* et le *profiling* sur une application espace utilisateur et comprendre ses interactions avec le noyau Linux afin de corriger des bugs, en utilisant *strace*, *ltrace*, *perf* ou *Callgrind*
- Être capable d'utiliser le *tracing* et le *profiling* le système Linux complet, en utilisant *perf*, *ftrace*, *kprobe*, les outils *eBPF*, *kernelshark* ou *LTTng*
- Être capable de débugger des problèmes au niveau du noyau Linux : debug de crash en direct ou post-mortem, analyse de problèmes mémoire au niveau noyau, analyse de problèmes de locks, utilisation de debuggers au niveau noyau.

Prérequis

- **Connaissance et pratique des commandes UNIX ou GNU/Linux** : les participants doivent être à l'aise avec l'utilisation de la ligne de commande Linux. Les participants manquant d'expérience sur ce sujet doivent se former par eux-mêmes, par exemple en utilisant nos supports de formation.
- **Expérience minimale en développement Linux embarqué** : les participants doivent avoir une compréhension minimale de l'architecture d'un système Linux embarqué : rôle du noyau Linux par rapport à l'espace utilisateur, développement d'applications espace utilisateur en C. Suivre la formation Linux embarqué de Bootlin permet de remplir ce pré-requis.
- **Niveau minimal requis en anglais : B1**, d'après le *Common European Framework of References for Languages*, pour nos sessions animées en anglais. Voir la grille CEFR pour une auto-évaluation.

Méthodes pédagogiques

- Présentations animées par le formateur : 40% de la durée de formation
- Travaux pratiques réalisés par les participants : 60% de la durée de formation
- Version électroniques de supports de présentation, des instructions et des données de travaux pratiques. Les supports sont librement disponibles ici.

Modalités d'évaluation

Seuls les participants qui auront assisté à l'intégralité des journées de formation, et qui auront obtenu plus de 50% de réponses correctes à l'évaluation finale recevront une attestation individuelle de formation de la part de Bootlin.

Handicap

Les participants en situation de handicap qui ont des besoins spécifiques sont invités à nous contacter à l'adresse *training@bootlin.com* afin de discuter des adaptations nécessaires à la formation.

Équipement nécessaire

Pour les sessions en présentiel délivrées chez nos clients, notre client devra fournir :

- Projecteur vidéo
- Un ordinateur sur chaque bureau (pour une ou deux personnes), avec au moins 16 Go de RAM et Ubuntu Linux 24.04 installé dans une partition dédiée d'au moins 30 Go.
- Les distributions autres que Ubuntu Linux 24.04 ne sont pas supportées, et l'utilisation de Linux dans une machine virtuelle n'est également pas supportée.
- Connexion à Internet rapide et sans filtrage : au moins 50 Mbit/s de bande passante en téléchargement, et pas de filtrage des sites Web et protocoles.
- Les ordinateurs contenant des données importantes doivent être sauvegardés avant d'être utilisés dans nos sessions.

Pour les sessions en présentiel organisés dans les locaux de Bootlin, Bootlin fournit l'ensemble de l'équipement.

Plateforme matérielle pour les travaux pratiques

Plateforme STM32MP1

Une de ces cartes de STMicroelectronics : **STM32MP157A-DK1**, **STM32MP157D-DK1**, **STM32MP157C-DK2** ou **STM32MP157F-DK2**

- Processeur STM32MP157, double Cortex-A7, de STMicroelectronics
- Alimentée par USB
- 512 Mo DDR3L RAM
- Port Gigabit Ethernet port
- 4 ports hôte USB 2.0
- 1 port USB-C OTG
- 1 connecteur Micro SD
- Debugger ST-LINK/V2-1 sur la carte
- Connecteurs compatibles Arduino Uno v3
- Codec audio
- Divers : boutons, LEDs
- Écran LCD tactile (uniquement sur cartes DK2)

BeaglePlay

Carte **BeaglePlay**

- SoC Texas Instruments AM625x (CPU 4xARM Cortex-A53)
- SoC avec accélération 3D, MCU intégré et de nombreux autres périphériques.
- 2 GB de RAM
- 16 Go de stockage eMMC
- USB hôte et device, microSD, HDMI
- WiFi 2.4 and 5 GHz, Bluetooth et aussi Ethernet
- 1 Header MicroBus (SPI, I2C, UART, ...), connecteurs OLDI et CSI.

Programme de la formation

Jour 1 - Matin

Cours	Pile logicielle Linux	▪ Vue d'ensemble : comprendre l'architecture général d'un système Linux, aperçu des principaux composants ▪ Différence entre un processus et un thread, comment les applications fonctionnent de façon concurrente. ▪ Fichiers ELF et outils d'analyse associés. ▪ Organisation de l'espace d'adressage des applications : heap, stack, bibliothèques partagées, etc. ▪ MMU et gestion mémoire : espaces d'adressage physique et virtuel ▪ Contexte d'exécution dans le noyau : threads noyau, workqueues, interruptions, interruptions threadées, softirq
Cours	Outils usuels d'analyse et d'observation	▪ Analyse d'un binaire ELF avec les outils GNU (*objdump*, *addr2line*) ▪ Outils pour monitorer un système Linux : processus, consommation et mapping mémoire, ressources ▪ Utilisation de *vmstat*, *iostat*, *ps*, *top*, *iotop*, *free* et compréhension des métriques qu'ils fournissent. ▪ Systèmes de fichiers virtuels : *procfs*, *sysfs* et *debugfs*

Jour 1 - Après-midi

TP	Comprendre ce qui fonctionne sur un système et sa charge	▪ Observation des processus en cours d'exécution avec *ps* et *top* ▪ Observation des mappings mémoire avec *procfs* et *pmap* ▪ Monitoring d'aurtres ressources avec *iostat*, *vmstat* et *netstat*
Cours	Debug d'une application	▪ Utilisation de *gdb* sur un processus en cours d'exécution. ▪ Comprendre l'impact des optimisations du compilateur sur la capacité à débugger un programme. ▪ Analyse post-mortem avec des fichiers *core* ▪ Debug à distance avec *gdbserver*. ▪ Étendre les capacités de *gdb* en utilisant des scripts Python.
TP	Résoudre un crash applicatif	▪ Analyse d'un code C compilé avec `compiler-explorer` pour comprendre les optimisations. ▪ Utilisation de *gdb* en ligne de commande, puis depuis un IDE. ▪ Utilisation des possibilités de scripting Python dans *gdb*. ▪ Debugger une application *post mortem* avec un *core dump* et *gdb*

Jour 2 - Matin

Cours	Tracing d'une application	▪ Tracing des appels systèmes avec *strace*. ▪ Tracing des appels à des bibliothèques partagées avec *ltrace*.
TP	Débugger des problèmes applicatifs	▪ Analyser les appels à des bibliothèques partagées d'une application en utilisant *ltrace*. ▪ Débugger une application qui fonctionne de manière incorrecte en utilisant *strace*.
Cours	Problèmes liés à la mémoire	▪ Problèmes classiques liés à la mémoire : *buffer overflow*, *segmentation fault*, fuite mémoire, collision pile/tas. ▪ Outils de détection/investigation de problèmes mémoires : *valgrind*, *libefence*, etc. ▪ Profiling de l'utilisation du tas en utilisant *Massif*

TP	Débugger des problèmes liés à la mémoire	▪ Fuites mémoire et détection de comportement incorrects avec *valgrind* et *vgdb*. ▪ Problèmes de performance liés à une sur-allocation. ▪ Visualisation de l'utilisation du tas par une application en utilisant *Massif*.

Jour 2 - Après-midi

Cours	Profiling d'application	▪ Problèmes de performance. ▪ Récupération d'informations de profiling avec *perf*. ▪ Analyse du graphe d'appel d'une application avec *Callgrind* et *KCachegrind*. ▪ Filtrage du jeu de données récupéré. ▪ Interprétation, des données enregistrées avec *perf*.
TP	Profiling d'application	▪ Profiling d'une application avec *Callgrind*/*KCachegrind*. ▪ Analyse des performances d'une application avec *perf*. ▪ Générer un *flamegraph* avec *FlameGraph*.

Jour 3 - Matin

Cours	Profiling et tracing de l'ensemble du système	▪ Profiling du système complet avec *perf*. ▪ Utilisation de *kprobes* pour ajouter des points de trace supplémentaires sans recompilation ▪ Tracing d'application et du noyau et visualisation des traces avec *ftrace*, *kernelshark* ou *LTTng* ▪ Tracing avec eBPF : principes généraux, développements d'outils avec BCC puis libbpf
TP	Profiling et tracing de l'ensemble du système	▪ Profiling du système complet avec *perf*. ▪ Analyse de latences système avec *ftrace* et *kernelshark*.
TP	Développement d'outils en eBPF	▪ Scripting python avec *BCC*. ▪ Développement d'un outil personnalisé de tracing avec libbpf.

Jour 3 - Après-midi

Cours	Debugging du noyau Linux	▪ Sorties de la compilation du noyau Linux utiles pour le debugging (`vmlinux`, `System.map`). ▪ Comprendre et configurer le comportement des *kernel oops*. ▪ Analyse post-mortem d'un crash kernel avec *crash*. ▪ Problèmes mémoire au niveau kernel (*KASAN*, *UBSAN*, *Kmemleak*). ▪ Debugging du noyau Linux avec *KGDB* et *KDB*. ▪ Options du noyau Linux pour le debug des problèmes de verrous (lockdep) ▪ Autres options de configuration du noyau Linux utiles pour le debug.
TP	Debugging du noyau Linux	▪ Analyse d'un *oops* après utilisation d'un module noyau incorrect, avec *obdjump* et *addr2line*. ▪ Debugging d'un *deadlock* avec les options *PROVE_LOCKING*. ▪ Détecter un *undefined behavior* avec *UBSAN* dans le noyau Linux. ▪ Trouver une fuite mémoire avec *kmemleak*. ▪ Débugger un module noyau avec *KGDB*.