Chapitre 6

Introduction Aux Systèmes Temps Réel

6.1. Définition et caractérisation d'un système temps réel

Ce chapitre inaugure la seconde partie du cours en établissant une transition fondamentale entre la programmation embarquée séquentielle classique et l'architecture orientée temps réel. La première section est consacrée à une définition précise du concept de système temps réel, que l'on peut formuler ainsi : il s'agit d'un système dont la correction ne dépend pas seulement des résultats logiques des calculs, mais également de l'instant auquel ces résultats sont produits. Cette définition conduit naturellement à une caractérisation essentielle qui distingue deux grandes catégories de contraintes temporelles. D'une part, les systèmes temps réel "dur" (hard real-time) sont ceux pour lesquels le non-respect d'une échéance temporelle constitue une défaillance critique pouvant entraîner des dommages matériels, des pertes humaines ou la destruction du système lui-même ; on trouve dans cette catégorie les systèmes de freinage automobile, les commandes de vol aéronautiques ou les dispositifs médicaux implantables. D'autre part, les systèmes temps réel "mou" (soft real-time) tolèrent des dépassements d'échéances occasionnels, ces retards n'entraînant qu'une dégradation progressive des performances sans conséquence catastrophique ; les systèmes de streaming multimédia, les interfaces utilisateur réactives ou les réseaux de capteurs domestiques illustrent typiquement cette seconde catégorie. Cette distinction est fondamentale car elle conditionne l'ensemble des choix architecturaux, des méthodes de validation et des marges de sécurité à appliquer lors de la conception.

6.2. Concepts fondamentaux : tâche, latence, gigue et déterminisme

La deuxième section de ce chapitre pose les concepts fondamentaux qui structurent la pensée des concepteurs de systèmes temps réel. La notion de **tâche** (ou processus léger) est d'abord introduite comme l'unité élémentaire d'exécution possédant sa propre logique fonctionnelle et ses propres contraintes temporelles ; une tâche se caractérise par son code exécutable, ses données propres, et surtout par ses paramètres temporels que sont sa période d'activation, son échéance (deadline) et son temps d'exécution maximal. Le concept de **latence** est ensuite développé comme l'intervalle de temps s'écoulant entre l'occurrence d'un événement déclencheur (par exemple, l'arrivée d'une interruption matérielle) et le début effectif du traitement associé par le système ; cette latence, qui résulte des délais d'interruption, de l'ordonnancement et des sections critiques, doit être bornée supérieurement pour garantir le respect des échéances. La **gigue** (jitter) est présentée comme la variation temporelle entre deux occurrences consécutives d'un même événement ou d'une même action ; dans un système temps réel bien conçu, la gigue doit être minimisée et parfaitement maîtrisée, car des variations imprévisibles compromettent le déterminisme des chaînes de traitement. Le **déterminisme** constitue le fil conducteur de cette section : un système temps réel se doit

d'être prévisible dans son comportement temporel, ce qui implique que, pour une charge donnée et des conditions d'entrée identiques, les temps de réponse doivent être reproductibles et bornés. Ces quatre concepts sont interdépendants et leur maîtrise conditionne la capacité à concevoir des systèmes répondant efficacement aux contraintes temps réel.

6.3. Architectures logicielles pour le temps réel

La troisième section de ce chapitre explore les différentes architectures logicielles permettant de répondre aux contraintes temporelles précédemment définies, en préparant ainsi le terrain pour l'étude approfondie des noyaux temps réel qui fera l'objet des chapitres suivants. L'approche la plus simple, dite **super-loop** ou boucle principale, consiste à structurer le programme comme une boucle infinie dans laquelle chaque fonction est appelée séquentiellement ; si cette architecture est facile à implémenter sur des microcontrôleurs de faible capacité, elle souffre d'un manque de réactivité face aux événements asynchrones et d'une absence de priorisation entre les différentes tâches. Une première amélioration est apportée par l'architecture dite **basée sur interruptions**, où les traitements critiques sont déportés dans les routines de service d'interruption (ISR) ; bien que cette approche améliore la réactivité, elle conduit rapidement à des difficultés de synchronisation et à des risques de blocages lorsque le nombre d'interruptions devient important. L'architecture la plus aboutie, qui sera le cœur de la suite du cours, est celle du **noyau temps réel** (RTOS), qui introduit un ordonnanceur capable de gérer plusieurs tâches concurrentes avec des priorités distinctes selon des politiques déterministes telles que l'ordonnancement préemptif à priorités fixes. Cette section présente également les principes généraux de l'**analyse d'ordonnançabilité**, qui permet de vérifier a priori, par des méthodes mathématiques comme l'analyse de charge processeur (utilization bound) ou le calcul du temps de réponse, que l'ensemble des tâches d'un système respectera ses échéances dans le pire cas de charge. Enfin, cette introduction aux architectures logicielles se conclut par une mise en perspective des compromis inhérents à la conception temps réel : complexité logicielle versus prédictibilité, consommation énergétique versus réactivité, et richesse fonctionnelle versus simplicité de validation. Ces compromis seront revisités en détail lors de l'étude pratique des RTOS dans les chapitres suivants.