Systeemarchitect bouwt besturingssysteem voor

De beste **realtime kernel** is niet te

De magie die van besturingssystemen uitgaat, heeft al van menig ontwikkelaar de zinnen geprikkeld. Als je het geluk hebt niet alleen met realtime besturingssystemen te werken, maar ook aan realtime besturingssystemen, kom je tal van creatieve bedenksels tegen. Zo zie je slimme oplossingen, maar ook al te vaak voorbeelden van hoe het niet moet. Bovendien kun je bij een gegeven kernel - om allerlei en vaak niet technische redenen - je ideeën niet optimaal implementeren.

KOOS HOLST Systeemarchitect, Turnkiek Technical Systems

Onlangs maakte ik - in een lopend project - kennis met een kernel die bijna tegen al mijn ideeën indruiste. Deze bleek verder na enige tijd uit het project te worden gehaald, omdat hij niet aan de verwachtingen voldeed. Vreemd genoeg bleek de gebruikte processor, een lid van de 16-bits 80C51XA-familie van Philips, voor een dergelijke kernel toch over bijna ideale eigenschappen te beschikken (zie kader: Eigenschappen van de 80C51XA-familie).
Toen was de tijd rijp om de uitdaging aan te gaan en te bewijzen dat het ook beter kan. Het resultaat is een compacte, efficiënte kernel met unieke, praktische eigenschappen. Bovendien is hij snel te leren, en in zeer korte tijd in de lucht.

Tekortkomingen bestaande producten

Een aantal tekortkomingen van bestaande kernels is algemeen van aard. Hieronder volgen er een paar.
Veel besturingssystemen proberen portable te zijn over meerdere processorarchitecturen. Dit kan een heel goede eigenschap zijn als dat voor de applicatie van belang is. Maar het probleem is dat menig kernelfabrikant dit vertaalt als: 'Voor elke implementatie is de sourcecode voor de kernel voor 99 % hetzelfde'. Dit houdt in dat er niet of nauwelijks gebruik wordt gemaakt van de specifieke eigenschappen van de onderliggende hardware. Dat is jammer, want dan wordt een belangrijk deel van de kracht van die processor weggelaten. Eigenlijk probeert zo'n kernelfabrikant zijn besparing aan inspanning te verkopen als een goede zaak voor de klant. Dat is op zijn minst bedenkelijk.
Verder wordt een kernel vaak gezien als extra software of zelfs overhead. Men zou verwachten dat de fabrikant er alles aan doet om die overhead te minimaliseren en zo zuinig mogelijk om te gaan met het geheugen. Maar het komt regelmatig voor dat in de applicatie delen van de kernel zitten die nooit gebruikt worden. Er hoort een mogelijkheid te zijn om tijdens het bouwen van de applicatie die delen automatisch weg te laten.
Een belangrijke eigenschap van realtime systemen is de gegarandeerde snelheid waarmee op externe signalen kan worden gereageerd (de worst case interrupt latency). Dat heeft te maken met zaken als: de snelheid van het interruptsysteem, de snelheid waarmee de context van het onderbroken programma kan worden veiliggesteld (registers en dergelijke), de kloksnelheid van de processor, de snelheid van het geheugen en de langst mogelijke ondeelbare instructie of instructiereeks.
Met name met betrekking tot dit laatste punt, vindt menig kernelfabrikant dat hij een (voor u negatieve) bijdrage moet leveren. In zo'n kernel gebeuren zaken die als een ondeelbare actie moeten plaatsvinden. Op zo'n moment neemt die kernelfabrikant de vrijheid het hele interruptsysteem even uit te schakelen. Dat betekent dat deze extra vertraging in de hoge prioriteit interruptroutine domweg geaccepteerd moet worden. Er is geen keuze; ook niet mits ..., met restricties als ...., enzo-

Het prioriteiten systeem van de XA-processor

De XA heeft een veelheid aan interruptvectoren. Elke interruptbron heeft zijn eigen vector zodat er daaromtrent in de interrupt serviceroutine weinig meer hoeft te worden uitgezocht.
De XA kent in zijn interruptsysteem programmeerbare prioriteitniveaus in een schaal van 0..15. De processor draait op zo'n niveau en elke interruptbron heeft zo'n niveau. Een actieve interrupt wordt alleen gehonoreerd als zijn prioriteit hoger is dan waarop de processor draait en er geen andere actieve interrupt is met een nog hoger niveau.
Het interrupt niveaumechanisme regelt dus niet alleen de belangrijkheid van de interrupts onderling, maar staat ook toe op een heel fijnmazige manier bepaalde interrupts (tijdelijk) uit te sluiten. Een kernel kan hier heel handig gebruik van maken om zijn critical sections te beschermen zonder tot het brute disable interrupts te hoeven overgaan.
Naast gebruikelijke trapinstructies heeft de XA ook software-interrupts. Dit zijn een soort externe interrupts die de XA zelf kan activeren. Hiermee kun je op heel elegante manier een nog te doen op het moment dat de processor prioriteit voldoende gezakt is -mechanisme maken. De applicatie kan hier heel handig gebruik van maken wanneer een interrupt gesplitst moet worden in een heel hoge prioriteit deel en een lagere prioriteit deel. Ook de kernel maakt gebruik van dit mechanisme voor scheduling en time-out support.

koop

voorts. De kernel degradeert de kwaliteit van het systeem.
Vervelend hierbij is dat een aantal processoren wel degelijk mogelijkheden biedt, om in die gevallen iets te doen waardoor de kwaliteit behouden blijft. Maar de meeste kernelbouwers wensen daar geen gebruik van te maken (zie kader: Het prioriteitensysteem van de XA-processor).
Ook bij de naamgeving van functies die de kernel ter beschikking stelt kunnen regelmatig kanttekeningen geplaatst worden. Wat te denken van een functie die zoiets als *qattb* heet. Er is hier niets gedaan om kernelnamen herkenbaar anders te maken dan die van de applicatie, waardoor onverwachte conflicten (of erger) kunnen ontstaan. Nu zijn er op dit gebied vele ideeën (zeg maar smaken), maar namen als *KernelProcessCreate* of *KernelTimerStart* zijn al een stuk prettiger leesbaar en eenvoudiger te onthouden.
Eén van de kwaliteiten van een kernel moet zijn dat de ontwikkeltijd van de applicatie wordt geminimaliseerd. Voor een eenvoudig hello world applicatietje zijn daarom dikke manuals, complexe configuratie-utilities en verwarrende architecturen uit den boze.
Aan de andere kant moet het ook niet zo zijn dat alle complexiteit wordt weggehouden door de gebruiker in een strak keurslijf te drukken, zodat die alleen maar kan doen wat de fabrikant voor hem heeft bedacht.
De documentatie zou zo moeten zijn opgezet dat, na het lezen van een overzichthoofdstuk, al binnen een uur het gevoel ontstaat de werking ongeveer onder de knie te hebben. Door bijgevoegde eenvoudige voorbeelden zou men in korte tijd een eenvoudig hello world applicatietje draaiend moeten kunnen krijgen. Als bovendien de opzet van de documentatie en de naamgeving van de functies zo is, dat al gauw het gevoel ontstaat dat: als die functie bestaat, zal die ongeveer zo of zo heten en kan het daar gevonden worden, zal iedereen snel met zo'n kernel overweg kunnen. De moeilijke en/of uitzonderlijke zaken moeten ook mogelijk zijn en goed beschreven staan met duidelijke indicatie waar de pijnpunten liggen. Eventueel mag dit laatste wat meer tijd kosten en moet duidelijk van de reguliere functies zijn gescheiden.

Het eisenpakket

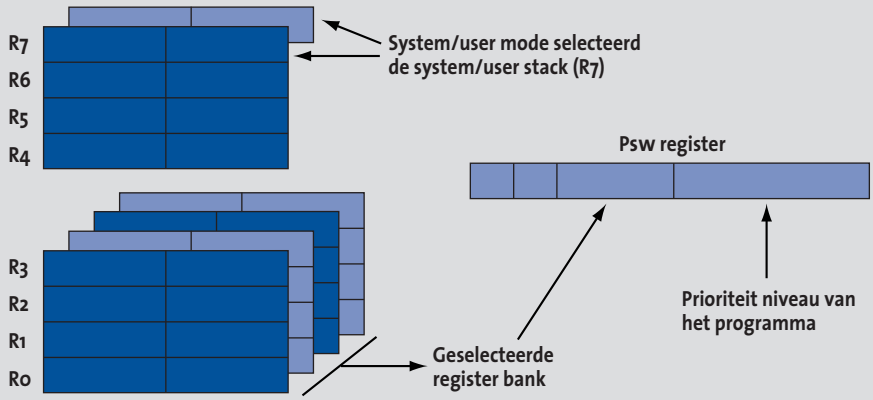
Maar hoe moet het dan wel? Allereerst worden de primitieven die de kernel beschikbaar moet stellen geformuleerd:

- dynamisch geheugenbeheer in de vorm van memory resources en regions;
- inter-process communicatiemechanismen, zoals semaforen en mailboxen;
- synchronisatiemechanismen, zoals timers en interrupts,
- time-out mechanismen en watchdog timers.

Tijdens het ontwerp vormde de volgende eisen de leidraad: ›

Eigenschappen van de 80C51XA-familie

De 80C51XA (of kortweg XA) is een 16-bits familie, met voor een 8-bits 8051-programmeur nogal wat herkenningspunten. Zo zijn daar de vertrouwde peripherals, de Harvard-architectuur en de special function registers. De instructieset van de 8051 is wel één op één te vertalen naar die van de XA, maar kan daarop niet rechtstreeks worden uitgevoerd. De XA is daardoor niet erfelijk belast met de beperkingen van zijn 8-bits broertje.
De XA heeft een lineaire 16 Mb coderuimte en een gesegmenteerde 16 Mb dataruimte. De omvang van de segmenten is vast en heeft een grootte van 64 kb. De ruimte voor special function registers is 1 kb (512 byte intern + 512 byte extern). Het bit-adresseerbare gebied is 1024 bits groot (512 bit in de special function registers, 256 bits in de register file en 256 bits in data memory).
De XAheeft 8 stuks 16-bit registers die ook als losse byteregisters gebruikt kunnen worden en waarop alle bewerkingen kunnen worden gedaan. De 16-bits registers kunnen ook gebruikt worden voor indirecte adressering in verschillende uitvoeringsvormen.
De instructieset is erg compleet en bevat ook multiply, divide en zelfs een paar 32-bits shiftoperaties. Ook de uit de 8051 bekende bitoperaties zijn aanwezig. Hogere programmeertalen kunnen efficiënt in de XA-instructieset worden vertaald.
De eerste helft van de registers is in viervoud uitgevoerd in de vorm van registerbanken. Dit kan een beperkt aantal interrupt serviceroutines enorm versnellen, omdat de keuze van de registerbank een onderdeel is van de interruptvector. Daardoor gebeurt dit overschakelen zonder extra overhead.
De XA kent de system en de user mode. De belangrijkste consequentie daarvan is dat er 2 stack pointers zijn waar een kernel efficiënt gebruik van kan maken. Ook de uitgebreide interruptstructuur van de XA levert een positieve bijdrage aan de configureerbaarheid en de efficiëntie van het systeem.
De XA is de generieke naam voor een familie van processoren. Zo zijn daar het 'basismodel' de XA-G3 (3 timers, 2 seriële poorten, RAM, ROM/EPROM), de XA-FLASH (flashgeheugen in plaats van ROM/EPROM), de XA-S3 (+I2C, +A/D, +PCA), SmartXA (smartcards), en de XA-SCC (Telecom). Bovendien is er nog een aantal in de maak.

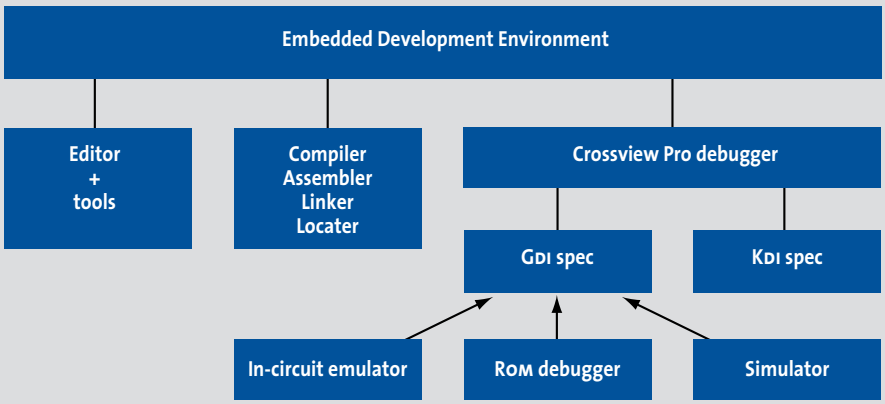


De ontwikkelomgeving

Als ontwikkelomgeving heb ik om de volgende redenen gekozen voor de Tasking Embedded Development Environment (EDE) voor de 80C51XA familie:

- **zeer krachtige en vriendelijke omgeving gebaseerd op Codewright die snel en efficiënt werkt. Bovendien zijn er instelmogelijkheden voor het geheugengebruik die heel mooi voor de kernel zijn te gebruiken waardoor een naadloos geheel ontstaat;**
- **een C-compiler die alle eigenschappen van de 80C51XA familie beschikbaar maakt en efficiënte en stabiele code aflevert;**
- **de manier waarop de C-compiler variabelen aan functies doorgeeft is efficiënt in het algemeen, maar past perfect bij de implementatie van de kernel;**
- **de beschikbaarheid van een zeer krachtige debugomgeving met een interface naar de onderliggende execution environment die volgens een open standaard (de GDI spec.) verloopt. Tasking levert zelf een instructiesimulator en een target (rom) debugger. Het is heel goed denkbaar dat iemand dat in de toekomst zelfs gaat doen voor een in-circuit emulator;**
- **de mogelijkheid om de debugomgeving 'kernel aware' te maken volgens de KDI-specificatie. En dat werkt dan met alle onderliggende execution environments.**

Het volgende plaatje geeft de structuur van die ontwikkelomgeving weer.



- geen limitatie op het aantal primitieven, anders dan die ontstaan door de processorsnelheid of hoeveelheid geheugen;
- zo kort mogelijke responsetijden voor interrupts en task switches;
- zo compact en snel mogelijke code;
- een heldere architectuur, die eenvoudig te begrijpen en te leren is en aansluit op de processorarchitectuur;
- zo goed mogelijk gebruikmaken van de eigenschappen van de 16-bits 80C51XA-processor en het voor de gebruiker beschikbaar stellen van die eigenschappen;
- de functionaliteit van de kernel moet vanuit de applicatie toegankelijk zijn via de C-programmeertaal.

Als ontwikkelomgeving is door de goede ervaringen en de juiste toolset gekozen voor de Embedded Development Environment (EDE) van Tasking voor de 80C51XA-familie. Dit bedrijf heeft goede papieren voor de XA-processorfamilie (zie kader Ontwikkelomgeving).

Het resultaat

Het resultaat, de kernel, laat inderdaad zien dat er bij bestaande producten nog ruimte voor verbetering is. Om op deze plaats alle details te geven gaat wat ver, daarom zullen we ons beperken tot een aantal in het oog springende zaken.

Scheduling en prioriteiten
De kernel heeft, zoals dat te verwachten is van een realtime kernel, een preemptive scheduling op basis van prioriteiten. Het aantal processen is onbeperkt en verschillende processen mogen desgewenst dezelfde prioriteit hebben. Wel is het aantal prioriteiten gelimiteerd op 256, waarbij er 1 gereserveerd is voor het idleproces van de kernel.

In dit idleproces kan naar keuze idletijd worden gemeten of stroom gespaard worden door de processor in de energiezuinige idle mode te schakelen. De 256 procesprioriteiten van de kernel zijn qua architectuur in lijn met de 16 interruptprioriteiten van de 80C51XA-processor.
Overigens is het aantal verschillende prioriteiten niet beperkt door de architectuur van de kernel, maar een keuze uit de praktijk. Desgewenst kan dit aantal worden verhoogd naar 65.536, of hoger.

Geheugengebruik
De hele kernel is geschreven als een verzameling van meer dan 100 modules, waarbij de linker (automatisch) bepaalt welke modules in de applicatie nodig zijn. In een gemiddelde applicatie kost de kernel ruim 2 kbyte code. In het ergste geval, als de hele kernel wordt gebruikt, kan dat oplopen tot een schamele 3 kbyte. Ook de hoeveelheid data voor de kernel is bescheiden en blijft bij vrijwel iedere applicatie onder de 100 bytes (70 tot 90 bytes is normaal).

Stackgebruik en interrupts
Elk proces heeft zijn eigen stack. Bij de bepaling van de grootte van die stack hoeft geen rekening te worden gehouden met het stackgebruik van interruptroutines. Daardoor is de minimum stackgrootte van 32 bytes voor een klein proces een reële mogelijkheid.
Door een juiste afstemming van de kernelarchitectuur op de onderliggende processorarchitectuur, kunnen alle interrupt- en systeemfuncties gebruikmaken van een gezamenlijke systeemstack, zonder dat dit extra executietijd kost.
Deze keuze verlaagt de totaal benodigde hoeveelheid geheugen en versnelt het hele systeem omdat de systeemstack in het interne geheugen van de processor kan worden geplaatst.

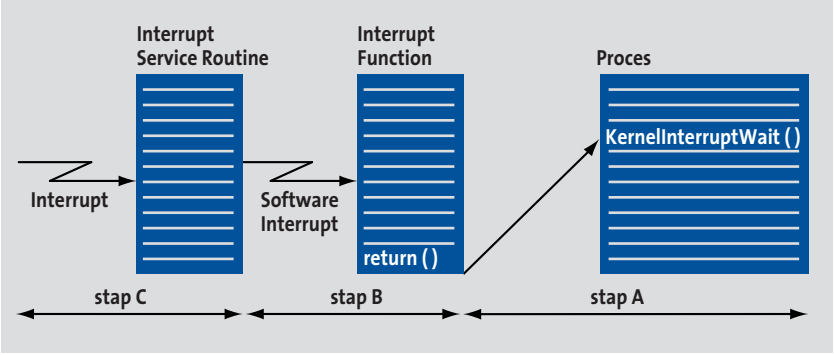
Interrupt latency
Het interruptsysteem kan uit meerdere stappen worden opgebouwd om een zo optimaal mogelijke oplossing voor de applicatie te bieden (zie figuur 1). Alle stappen zijn optioneel.
Stap A is dat een proces wacht op een

interrupt. Als die interrupt komt zal dat proces via het schedulingmechanisme worden geactiveerd en kan het, met gebruikmaking van alle kernelfunctionaliteit en eventueel andere processen, de interrupt afhandelen. Deze luxe betaal je met een relatief hoge latency omdat er minimaal een volledige taskswitch gedaan moet worden en processen met hogere prioriteit gewoon voorgaan.

Stap B is de interruptfunctie. Het kost totaal 8 XA-instructies aan kerneloverhead, plus veiligstellen van registers, voordat de executie in de interruptfunctie aangeland is. Het is mogelijk dat de kernel hier nog enige latency aan toevoegt omdat de kernel met een ondeelbare actie bezig is. Ook kan een interrupt van hogere prioriteit de zaak nog ophouden.

De interrupt functie kan met zijn returnwaarde aangeven of hij een eventueel wachtend process (stap A) wel of niet wil activeren. Zo is het bijvoorbeeld mogelijk een interruptfunctie verbonden aan een seriële poort, een hele regel tekst binnen te laten halen alvorens hij een wachtend proces wakker maakt.

Stap C is een interrupt-serviceroutine met een dusdanig hoge prioriteit dat hij zelfs boven de kernel staat en dus altijd direct wordt bediend. Wel moet die routine zelf de gebruikte registers veiligstellen, maar daar heeft de 80C51XA-processor ook een trucje voor (registerbanken) dat geen overhead kost. Dit is de kortst mogelijke latency, denkbaar op deze processor (1,53 µs worst case bij 30 Mhz). Het is de interrupt-serviceroutine verboden om iets met de kernel te doen. Het is echter wel mogelijk via een handige eigenschap van de processor een interruptfunctie (stap B) of een proces (stap A) te activeren.



1. Het interruptsysteem kan uit meerdere stappen worden opgebouwd, om een zo optimaal mogelijke oplossing voor de applicatie te bieden, waarbij alle stappen optioneel zijn.

Door een combinatie van deze stappen is het dus mogelijk in stap C kortstondig beschikbare data aan te pakken, in stap B daar enige intelligentie aan toe te voegen (bijvoorbeeld een communicatieprotocol) en in stap A de data te verwerken (bijvoorbeeld ontvangen commando's uitvoeren). De kernel introduceert hier dus geen degradatie van de worst case latency.

Timers

Het is mogelijk om, afgeleid van 1 hardware timer, een ongelimiteerd aantal software timers te maken. De structuur komt erg overeen met die van de interrupt. Zo is daar een timerfunctie (= interruptfunctie), en een proces kan op een timer wachten. Bij het starten van de timer kan de initiële tijd en de optionele herlaadtijd worden opgegeven, waardoor eenmalige of continu lopende timers mogelijk zijn. Het is zelfs mogelijk om de herlaadtijd *on-the-fly* in de timerfunctie te wijzigen, waardoor zeer complexe timerreeksen met hoge nauwkeurigheid mogelijk zijn.

Time-out support en watchdogs

Bij elke wachtende functie (bijvoorbeeld het wachten op een interrupt) kan, optioneel, een *time-out* worden gespecificeerd. Hierdoor kan worden ingespeeld op situaties die nooit of te laat plaatsvinden. Dit kan op fouten duiden of juist onderdeel zijn van een bepaald protocol.

Ook is het mogelijk softwarefouten met watchdog timers te signaleren. Vaak is zo'n watchdog timer al in hardware aanwezig, maar voldoet die niet altijd in multitaskingsystemen. Zo kan een systeem nog min of meer doordraaien en de hardware watchdog tevreden stellen, terwijl er een proces met niet zo'n hoge prioriteit is dat in een oneindige lus vastzit. De watchdogfunctie van de kernel kan dit in veel gevallen wel onderkennen omdat hij per proces een aparte watchdog beschikbaar heeft. Bij het signaleren van zo'n fout kan de applicatiesoftware daar op een meer intelligente manier mee omgaan dan bijvoorbeeld de hardware watchdog door informatie daarover te loggen.

Zowel de time-out als de watchdog support kunnen uit de kernel weggelaten worden om ruimte en overhead te sparen.

Proceshiërarchie en threads

Door eigendom en gebruiksrechten aan resources (zoals semaforen, interrupts, timers, geheugen en processen) toe te kennen, kunnen systemen en subsystemen op een hiërarchische manier worden samengesteld. Dit heeft als voordeel dat de resources niet misbruikt kunnen worden door ongerelateerde processen of subsystemen. Ook bevordert het de modulariteit en herbruikbaarheid van die software. Dat is niet gratis, maar kost een heel klein beetje overhead. Mocht dat te veel zijn, dan kan dat voor de uiteindelijke geteste applicatie uit de kernel worden weglaten.

Door deze hiërarchie kunnen processen worden gemaakt die qua functionaliteit zo op threads lijken dat de expliciete aanwezigheid van threads niet meer nodig is.

Conclusie

Om diverse redenen zou je zelf een kernel willen maken. Bij de hier beschreven kernel was onvrede over bestaande producten de trigger voor de uitdaging. Hoewel het eigenlijk om die uitdaging ging, is er toch een volwaardige kernel met unieke eigenschappen ontstaan èn de wetenschap dat niet zo maar alles van die kernelbouwers voor zoete koek moet worden aangenomen - alhoewel, ik ben er nu zelf ook een! □

Meer informatie

Voor meer informatie over de kernelcode, een users manual of commentaar op het artikel kunt u contact opnemen met de auteur op holst@turnkiek.nl (www.turnkiek.nl).
Meer informatie over de XA-familie: www.semiconductors.philips.com en de ontwikkelomgeving: www.tasking.com